Prac2

October 10, 2020

1 Pràctica 2: La regressió

1.1 Objectius

Els objectius d'aquesta pràctica són: * Aplicar models de regressió, ficant l'èmfasi en: * Analitzar els atributs per seleccionar els més representatius i normalitzar-los. * Avaluar correctament l'error del model * Visualitzar les dades i el model resultant * Saber aplicar el procès de descens del gradient

- Ésser capaç d'aplicar tècniques de regressió en casos reals
- Validar els resultats en dades reals
- Fomentar la capacitat per presentar resultats tècnics d'aprenentatge computacional de forma adequada davant altres persones

1.2 Avaluació i entregues

1.2.1 Entregues

Les pràctiques es realitzarán els divendres de 15:00 a 17:00. Cada setmana presentarem nous mètodes i algorismes vistos a la classe de teoria.

L'entrega, es realitzarà el següent dijous abans de les 23:59, és a dir, tindreu les hores de classe més tota una setmana per a realitzar les tasques. Es pujará un .ipynb ja executat, on no només hi hagi codi sinó també text explicatiu de gràfiques, resultats, i analisis. Tingueu en compte que no es valora tant la quantitat, sino la qualitat del que s'hi explica. Hem d'aprendre a extreure el gra de la palla i presentar-ho de forma correcta i concisa.

1.2.2 Avaluació

Generalment, per tal de que veigueu varietat de dades i problematiques, es treballarà sobre dues bases de dades cada setmana. La primera serà comuna per tothom i, en aquest cas, tindrá la temática del COVID. La segona, tot i ser de lliure elecció (us deixem que l'escolliu vosaltres mateixos), haurà de tenir uns mínims de dificultat (o almenys, que tinguin una mica de gràcia).

COVID data Durant el primer mes, treballarem amb una base de dades comuna, amb la temática del COVID-19. Podeu trobar la darrera actualització a OWID DATA, tot i que no farem servir cap dada a partir del 1 de Octubre de 2020, per això us recomanem de fer servir la que tenim penjada al Campus Virtual.

Dataset lliure elecció Podeu trobar datasets de dades a qualsevol d'aquestes dues webs: * https://www.kaggle.com/datasets?search=machine+learning * https://archive.ics.uci.edu/ml/datasets.php

L'objectiu será escollir un dataset adequat per la tasca que volem realitzar. LLavors, el primer apartat serà conèixer la base de dades que es té entre mans. S'han d'analitzar els diferents atributs que la composen, entendre'ls i, si no està estipulat, **caldrà fixar quin es l'atribut objectiu a predir de tots els que hi ha a la base de dades**, justificant el per què de la decisió (és útil i representatiu pel problema, per exemple, donat un conjunt de dades sobre persones: edat, gènere, pes, alçada, risc de patir càncer, aquesta última pot ser justificada com la de més interés). També podeu mirar que l'atribut objectiu tingui valors que canvien. Per exemple, no té sentit predir un atribut on el 99% dels valors són 0, i hi ha algun 1.

1.2.3 Pràctica 2: Regressió Lineal i Polinomial

En la pràctica 1, es presenten diversos problemes per comprendre els mètodes de regressió numèrica. Cada apartat pot tenir una qualificació diferent.

- A. Exploració de la base de dades (30%)
- B. Ús de regressor Lineal (30%)
- C. Demostració d'assoliment (A+B en una nova BBDD) (20%)
- D. Implementació Regressor Lineal (20%)
- (Extra) Implementació Regresor polinomial (+10%, max: 10pts)

A continuació us posem un esquelet de codi que podeu tenir com a referència (o no)

[5]:
```
# Codi esquelet per la pràctica de la 2 sessió: Regressió Lineal i Polinomial

import numpy as np
import pandas as pd
%matplotlib inline
from matplotlib import pyplot as plt
import seaborn as sns
import datetime



# Visualitzarem només 3 decimals per mostra, i definim el num de files i␣
 ↪columnes a mostrar
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', 50)

# Funcio per a llegir dades en format csv
def load_dataset(path):
    dataset = pd.read_csv(path, header=0, delimiter=',')
    return dataset

# Carreguem dataset d'exemple
```

```
dataset = load_dataset('owid-covid-data.csv')

print("Dimensionalitat de la BBDD:", dataset.shape)
```

```
Dimensionalitat de la BBDD: (47328, 41)
```

A continuació veurem algunes taules i gràfiques per entendre com són les dades que tenim:

[6]:
```
dataset.describe()
```

[6]:

	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths
count	46714.000	46502.000	45720.000	46714.000	46502.000
mean	105304.798	1463.590	1451.104	4181.072	43.656
std	1089825.655	13268.625	13077.552	38853.696	367.208
min	0.000	-8261.000	-552.000	0.000	-1918.000
25%	62.000	0.000	0.571	1.000	0.000
50%	1067.000	10.000	15.143	21.000	0.000
75%	11496.000	168.000	173.465	248.000	3.000
max	34029923.000	321127.000	297041.143	1015043.000	10491.000

	new_deaths_smoothed	total_cases_per_million	new_cases_per_million
count	45720.000	46438.000	46438.000
mean	43.712	1967.496	25.122
std	357.460	4142.360	75.623
min	-232.143	0.000	-2212.545
25%	0.000	32.357	0.000
50%	0.143	290.308	1.594
75%	3.143	2047.809	18.209
max	7456.714	43650.601	4944.376

	new_cases_smoothed_per_million	total_deaths_per_million
count	45655.000	46438.000
mean	24.747	58.924
std	58.381	144.982
min	-269.978	0.000
25%	0.149	0.026
50%	3.014	5.114
75%	20.044	36.511
max	882.924	1237.551

	new_deaths_per_million	new_deaths_smoothed_per_million	new_tests
count	46438.000	45655.000	17022.000
mean	0.568	0.567	23599.478
std	2.993	1.900	96057.307
min	-67.901	-9.678	-3743.000
25%	0.000	0.000	991.250
50%	0.000	0.021	3333.500
75%	0.206	0.297	12178.250

```
max                  215.382                          63.140 1492409.000

       total_tests  total_tests_per_thousand  new_tests_per_thousand  \
count     17430.000                 17430.000               17022.000
mean    1572278.393                    55.184                   0.745
std     6965831.413                   110.468                   1.432
min           1.000                     0.000                  -0.398
25%       47101.250                     2.755                   0.062
50%      198537.000                    14.550                   0.292
75%      777617.000                    61.396                   0.869
max   113263096.000                  1327.460                  25.920

      new_tests_smoothed  new_tests_smoothed_per_thousand  tests_per_case  \
count           19196.000                        19196.000       17665.000
mean            22774.285                            0.729         173.259
std             87943.651                            1.295         887.276
min                 0.000                            0.000           1.488
25%              1066.000                            0.063          11.656
50%              3723.500                            0.307          32.850
75%             13095.500                            0.870         106.647
max           1169107.000                           19.058       47299.000

      positive_rate  stringency_index     population  population_density  \
count      18098.000         39364.000      47052.000           44901.000
mean           0.065            57.369   88300500.969             360.499
std            0.091            27.398  612410778.511            1654.659
min            0.000             0.000        809.000               0.137
25%            0.008            38.890    1399491.000              37.728
50%            0.029            62.960    8654618.000              88.125
75%            0.084            79.630   31072945.000             214.243
max            0.672           100.000 7794798729.000           19347.500

      median_age  aged_65_older  aged_70_older  gdp_per_capita  \
count   42195.000      41567.000      41976.000       41649.000
mean       31.319          9.256          5.854       20893.575
std         9.027          6.318          4.315       20427.288
min        15.100          1.144          0.526         661.240
25%        23.300          3.552          2.142        5338.454
50%        31.400          6.981          4.419       14103.452
75%        39.700         14.762          9.473       32415.132
max        48.200         27.049         18.493      116935.600

      extreme_poverty  cardiovasc_death_rate  diabetes_prevalence  \
count       27790.000              42203.000            43693.000
mean           12.127                251.593                8.049
std            19.229                117.530                4.148
min             0.100                 79.370                0.990
```

```
25%              0.500               155.898               5.310
50%              1.900               238.339               7.110
75%             16.000               318.991              10.180
max             77.600               724.417              23.360

       female_smokers  male_smokers  handwashing_facilities  \
count       33054.000     32635.000               19766.000
mean           10.811        32.635                  52.445
std            10.479        13.420                  31.603
min             0.100         7.700                   1.188
25%             1.900        21.400                  21.222
50%             6.400        31.400                  55.182
75%            19.600        40.900                  83.741
max            44.000        78.100                  98.999

       hospital_beds_per_thousand  life_expectancy  human_development_index
count                   38120.000        46458.000                40717.000
mean                        3.110           74.023                    0.725
std                         2.526            7.373                    0.153
min                         0.100           53.280                    0.354
25%                         1.300           69.910                    0.606
50%                         2.500           75.445                    0.754
75%                         4.200           79.380                    0.853
max                        13.800           86.750                    0.953
```

[7]:
```
covid_spain = dataset[dataset.location=="Spain"]
covid_italy = dataset[dataset.location=="Italy"]
covid_spain
```

[7]:
```
      iso_code continent location        date  total_cases  new_cases  \
13450      ESP    Europe    Spain  2019-12-31        0.000      0.000
13451      ESP    Europe    Spain  2020-01-01        0.000      0.000
13452      ESP    Europe    Spain  2020-01-02        0.000      0.000
13453      ESP    Europe    Spain  2020-01-03        0.000      0.000
13454      ESP    Europe    Spain  2020-01-04        0.000      0.000
…          …      …         …      …          …           …          …
13720      ESP    Europe    Spain  2020-09-26   716481.000      0.000
13721      ESP    Europe    Spain  2020-09-27   716481.000      0.000
13722      ESP    Europe    Spain  2020-09-28   748266.000  31785.000
13723      ESP    Europe    Spain  2020-09-29   758172.000   9906.000
13724      ESP    Europe    Spain  2020-09-30   769188.000  11016.000

       new_cases_smoothed  total_deaths  new_deaths  new_deaths_smoothed  \
13450                 nan         0.000       0.000                  nan
13451                 nan         0.000       0.000                  nan
13452                 nan         0.000       0.000                  nan
13453                 nan         0.000       0.000                  nan
```

```
13454                nan         0.000       0.000                nan
…                    …           …           …                    …
13720          10920.143     31232.000       0.000            105.286
13721          10920.143     31232.000       0.000            105.286
13722          10971.143     31411.000     179.000            106.857
13723          10843.571     31614.000     203.000            101.429
13724          10804.571     31791.000     177.000            108.143

       total_cases_per_million  new_cases_per_million  \
13450                    0.000                  0.000
13451                    0.000                  0.000
13452                    0.000                  0.000
13453                    0.000                  0.000
13454                    0.000                  0.000
…                        …                      …
13720                15324.229                  0.000
13721                15324.229                  0.000
13722                16004.052                679.823
13723                16215.924                211.871
13724                16451.536                235.612

       new_cases_smoothed_per_million  total_deaths_per_million  \
13450                             nan                     0.000
13451                             nan                     0.000
13452                             nan                     0.000
13453                             nan                     0.000
13454                             nan                     0.000
…                                 …                       …
13720                         233.562                   667.996
13721                         233.562                   667.996
13722                         234.653                   671.824
13723                         231.924                   676.166
13724                         231.090                   679.952

       new_deaths_per_million  new_deaths_smoothed_per_million  new_tests  \
13450                   0.000                              nan        nan
13451                   0.000                              nan        nan
13452                   0.000                              nan        nan
13453                   0.000                              nan        nan
13454                   0.000                              nan        nan
…                       …                                  …          …
13720                   0.000                            2.252   94782.000
13721                   0.000                            2.252   64576.000
13722                   3.828                            2.285        nan
13723                   4.342                            2.169        nan
13724                   3.786                            2.313        nan
```

```
       total_tests  total_tests_per_thousand  new_tests_per_thousand  \
13450          nan                       nan                     nan
13451          nan                       nan                     nan
13452          nan                       nan                     nan
13453          nan                       nan                     nan
13454          nan                       nan                     nan
…              …                         …                       …
13720  9533306.000                   203.900                   2.027
13721  9597882.000                   205.281                   1.381
13722          nan                       nan                     nan
13723          nan                       nan                     nan
13724          nan                       nan                     nan

       new_tests_smoothed  new_tests_smoothed_per_thousand  tests_per_case  \
13450                 nan                              nan             nan
13451                 nan                              nan             nan
13452                 nan                              nan             nan
13453                 nan                              nan             nan
13454                 nan                              nan             nan
…                     …                                …               …
13720          105165.000                            2.249           9.630
13721           99074.000                            2.119           9.073
13722                 nan                              nan             nan
13723                 nan                              nan             nan
13724                 nan                              nan             nan

       positive_rate      tests_units  stringency_index   population  \
13450            nan              NaN               nan 46754783.000
13451            nan              NaN             0.000 46754783.000
13452            nan              NaN             0.000 46754783.000
13453            nan              NaN             0.000 46754783.000
13454            nan              NaN             0.000 46754783.000
…                …                …               …           …
13720          0.104  tests performed            55.090 46754783.000
13721          0.110  tests performed            55.090 46754783.000
13722            nan              NaN            55.090 46754783.000
13723            nan              NaN               nan 46754783.000
13724            nan              NaN               nan 46754783.000

       population_density  median_age  aged_65_older  aged_70_older  \
13450              93.105      45.500         19.436         13.799
13451              93.105      45.500         19.436         13.799
13452              93.105      45.500         19.436         13.799
13453              93.105      45.500         19.436         13.799
13454              93.105      45.500         19.436         13.799
…                  …           …              …              …
13720              93.105      45.500         19.436         13.799
```

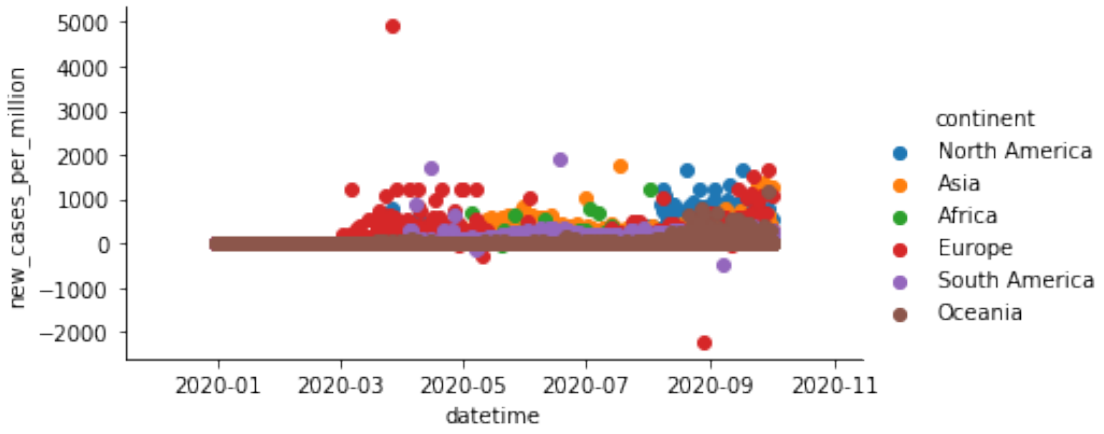
```
13721              93.105      45.500         19.436         13.799
13722              93.105      45.500         19.436         13.799
13723              93.105      45.500         19.436         13.799
13724              93.105      45.500         19.436         13.799

       gdp_per_capita  extreme_poverty  cardiovasc_death_rate  \
13450        34272.360            1.000                 99.403
13451        34272.360            1.000                 99.403
13452        34272.360            1.000                 99.403
13453        34272.360            1.000                 99.403
13454        34272.360            1.000                 99.403
…                …                …                      …
13720        34272.360            1.000                 99.403
13721        34272.360            1.000                 99.403
13722        34272.360            1.000                 99.403
13723        34272.360            1.000                 99.403
13724        34272.360            1.000                 99.403

       diabetes_prevalence  female_smokers  male_smokers  \
13450                7.170          27.400        31.400
13451                7.170          27.400        31.400
13452                7.170          27.400        31.400
13453                7.170          27.400        31.400
13454                7.170          27.400        31.400
…                    …              …             …
13720                7.170          27.400        31.400
13721                7.170          27.400        31.400
13722                7.170          27.400        31.400
13723                7.170          27.400        31.400
13724                7.170          27.400        31.400

       handwashing_facilities  hospital_beds_per_thousand  life_expectancy  \
13450                     nan                       2.970           83.560
13451                     nan                       2.970           83.560
13452                     nan                       2.970           83.560
13453                     nan                       2.970           83.560
13454                     nan                       2.970           83.560
…                         …                         …               …
13720                     nan                       2.970           83.560
13721                     nan                       2.970           83.560
13722                     nan                       2.970           83.560
13723                     nan                       2.970           83.560
13724                     nan                       2.970           83.560

       human_development_index
13450                    0.891
13451                    0.891
```

```
13452                    0.891
13453                    0.891
13454                    0.891
…                        …
13720                    0.891
13721                    0.891
13722                    0.891
13723                    0.891
13724                    0.891

[275 rows x 41 columns]
```

[8]:
```
dataset["datetime"] = pd.to_datetime(dataset["date"], format='%Y-%m-%d',␣
↪errors='ignore')
```

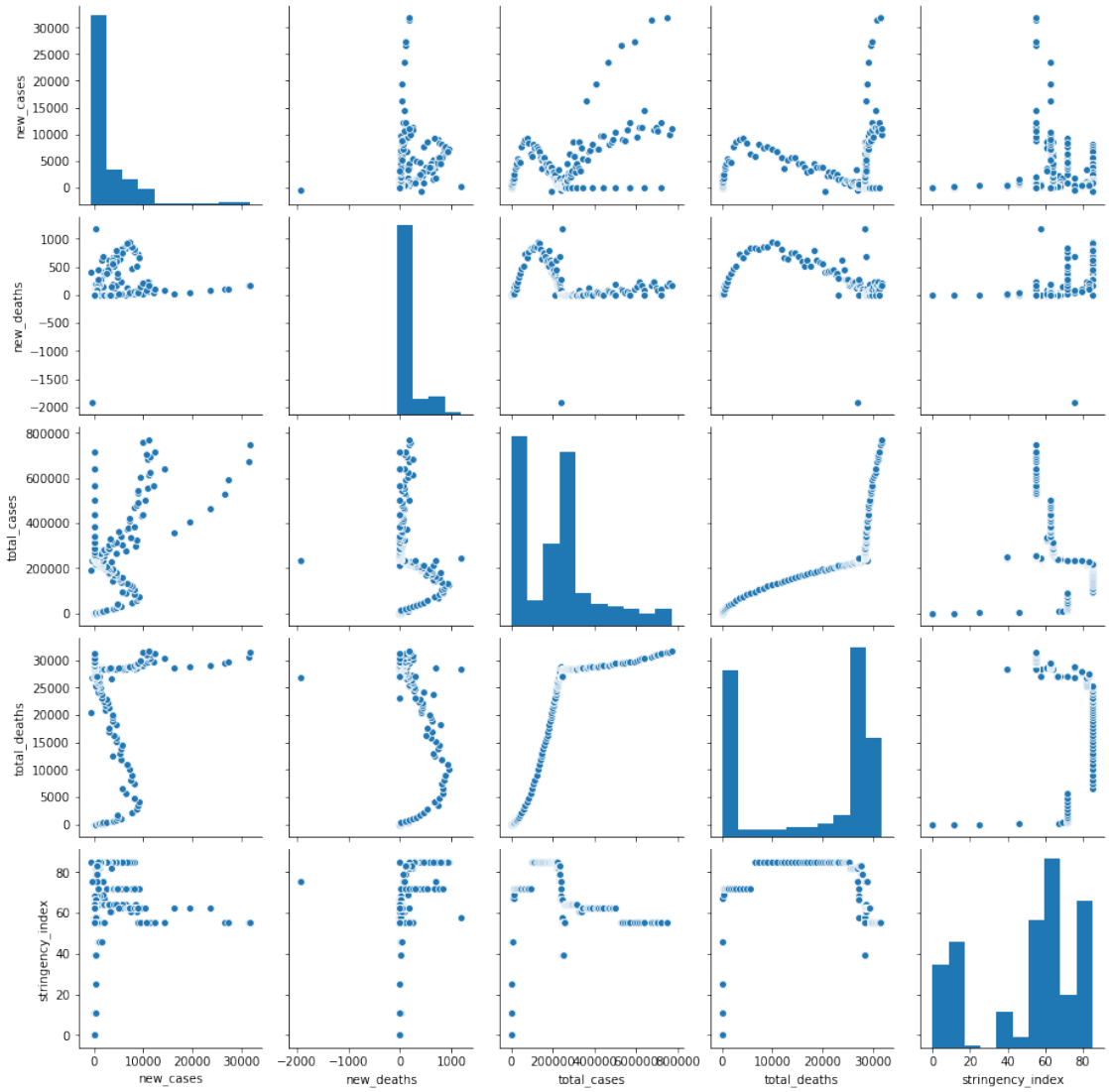
[9]:
```
fg = sns.FacetGrid(data=dataset, hue='continent',aspect=2)
fg.map(plt.scatter, 'datetime', 'new_cases_per_million').add_legend()
```

[9]: <seaborn.axisgrid.FacetGrid at 0x7fdf9ea9c050>



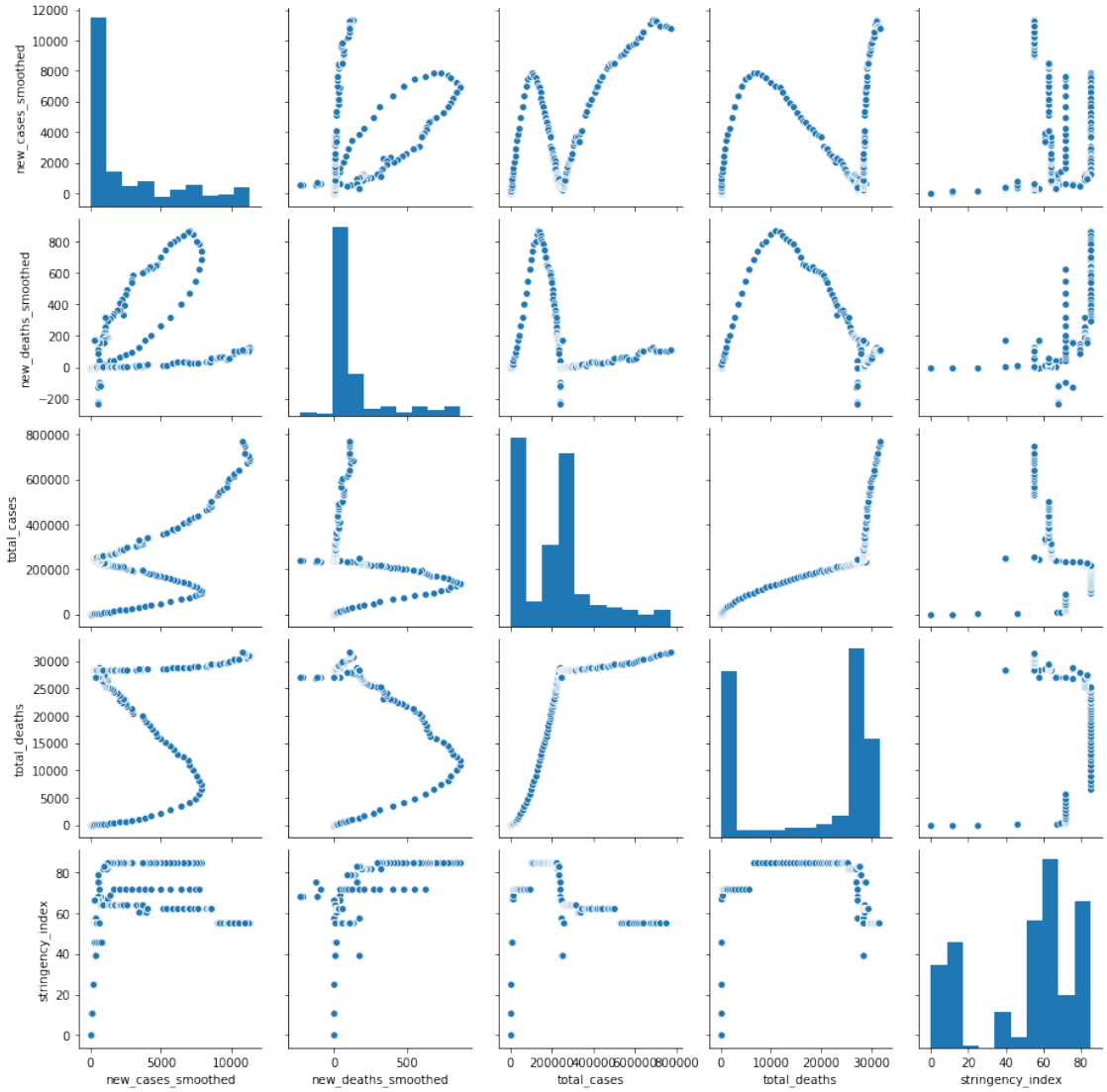
[10]:
```
print("Quina correlació entre aquests atributs hi veieu?")

selected_columns = ["location", "new_cases", "new_deaths", "total_cases",␣
↪"total_deaths", "stringency_index"]
sns.pairplot(covid_spain[selected_columns])
plt.show()
```

```
Quina correlació entre aquests atributs hi veieu?
```



[11]:
```
print("Què passa si utilitzem les dades smoothed?")

selected_columns = ["location", "new_cases_smoothed", "new_deaths_smoothed",␣
 ↪"total_cases", "total_deaths", "stringency_index"]
sns.pairplot(covid_spain[selected_columns])
plt.show()
```

```
Què passa si utilitzem les dades smoothed?
```



[12]:
```
print("I si busquem alguna correlació entre paisos per un dia concret?")

covid_agost = dataset[dataset.date=="2020-08-01"]
covid_agost
```

```
I si busquem alguna correlació entre paisos per un dia concret?
```

[12]:

	iso_code	continent	location	date	total_cases	\
136	ABW	North America	Aruba	2020-08-01	121.000	
412	AFG	Asia	Afghanistan	2020-08-01	36710.000	
606	AGO	Africa	Angola	2020-08-01	1109.000	
795	AIA	North America	Anguilla	2020-08-01	3.000	
1002	ALB	Europe	Albania	2020-08-01	5276.000	

```
…         …            …             …          …           …
46322       ZAF         Africa   South Africa  2020-08-01   493183.000
46519       ZMB         Africa         Zambia  2020-08-01     5963.000
46714       ZWE         Africa       Zimbabwe  2020-08-01     3169.000
46990  OWID_WRL            NaN          World  2020-08-01 17577428.000
47266       NaN            NaN  International  2020-08-01      696.000

       new_cases  new_cases_smoothed  total_deaths  new_deaths  \
136        1.000               0.429         3.000       0.000
412      168.000              96.286      1283.000      12.000
606       31.000              36.857        51.000       3.000
795        0.000               0.000         0.000       0.000
1002      79.000             100.857       157.000       3.000
…            …                   …             …           …
46322  11014.000           10169.571      8005.000     193.000
46519    408.000             301.000       151.000       2.000
46714     77.000             124.714        67.000      14.000
46990 279062.000          259257.000    674453.000    6123.000
47266        nan                 nan         7.000         nan

       new_deaths_smoothed  total_cases_per_million  new_cases_per_million  \
136                  0.000                 1133.320                  9.366
412                  5.286                  943.015                  4.316
606                  2.571                   33.743                  0.943
795                  0.000                  199.973                  0.000
1002                 4.143                 1833.345                 27.452
…                        …                       …                      …
46322              237.429                 8315.527                185.706
46519                3.000                  324.359                 22.193
46714                5.000                  213.215                  5.181
46990             5666.000                 2255.020                 35.801
47266                  nan                      nan                    nan

       new_cases_smoothed_per_million  total_deaths_per_million  \
136                             4.014                    28.099
412                             2.473                    32.958
606                             1.121                     1.552
795                             0.000                     0.000
1002                           35.047                    54.556
…                                 …                         …
46322                         171.468                   134.972
46519                          16.373                     8.214
46714                           8.391                     4.508
46990                          33.260                    86.526
47266                             nan                       nan

       new_deaths_per_million  new_deaths_smoothed_per_million  new_tests  \
```

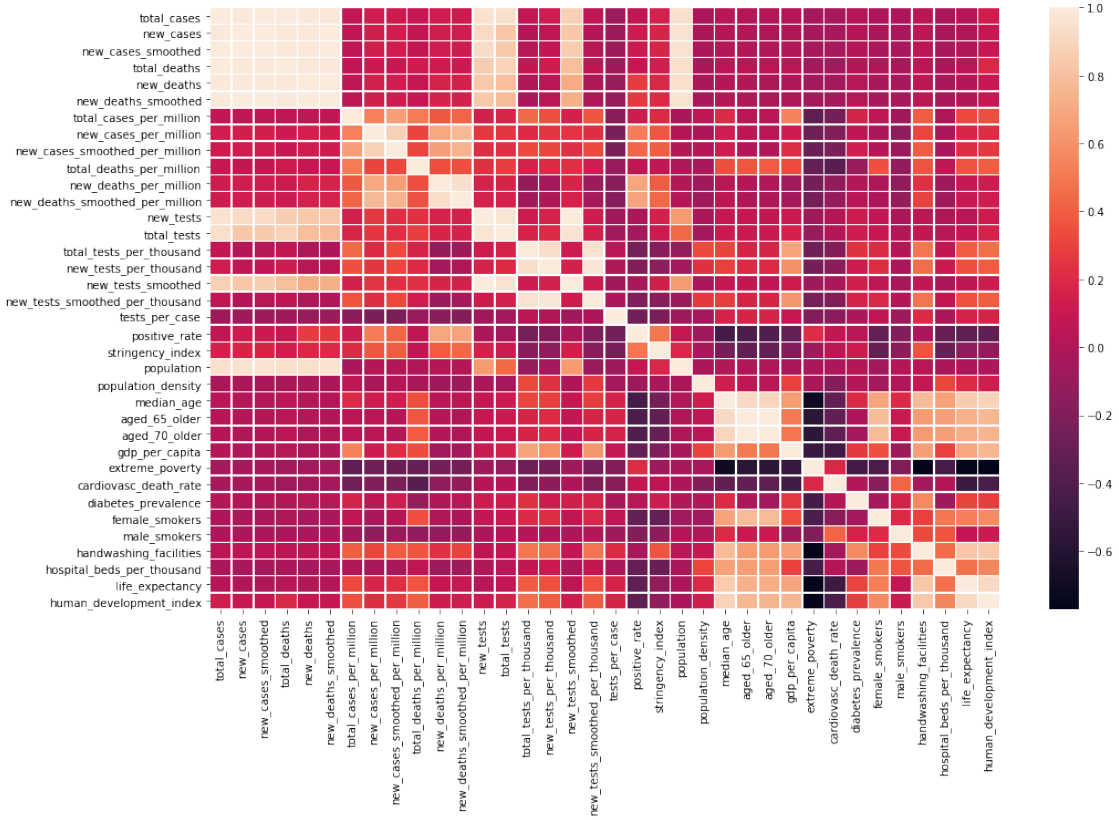
```
136                   0.000                          0.000        nan
412                   0.308                          0.136        nan
606                   0.091                          0.078        nan
795                   0.000                          0.000        nan
1002                  1.042                          1.440        nan
…                     …                              …        …
46322                 3.254                          4.003  42450.000
46519                 0.109                          0.163   1908.000
46714                 0.942                          0.336   1445.000
46990                 0.786                          0.727        nan
47266                   nan                            nan        nan

       total_tests  total_tests_per_thousand  new_tests_per_thousand  \
136            nan                       nan                     nan
412            nan                       nan                     nan
606            nan                       nan                     nan
795            nan                       nan                     nan
1002           nan                       nan                     nan
…              …                         …                       …
46322  3001985.000                    50.616                   0.716
46519    85001.000                     4.624                   0.104
46714    61577.000                     4.143                   0.097
46990          nan                       nan                     nan
47266          nan                       nan                     nan

       new_tests_smoothed  new_tests_smoothed_per_thousand  tests_per_case  \
136                   nan                              nan             nan
412                   nan                              nan             nan
606                   nan                              nan             nan
795                   nan                              nan             nan
1002                  nan                              nan             nan
…                     …                                …               …
46322           38739.000                            0.653           3.809
46519            1276.000                            0.069           4.239
46714            1367.000                            0.092          10.961
46990                 nan                              nan             nan
47266                 nan                              nan             nan

       positive_rate      tests_units  stringency_index     population  \
136              nan              NaN            32.410     106766.000
412              nan              NaN            78.700   38928341.000
606              nan              NaN            79.170   32866268.000
795              nan              NaN            24.070      15002.000
1002             nan              NaN            59.260    2877800.000
…                …                …              …             …
46322          0.263    people tested            80.560   59308690.000
46519          0.236  tests performed            50.930   18383956.000
```

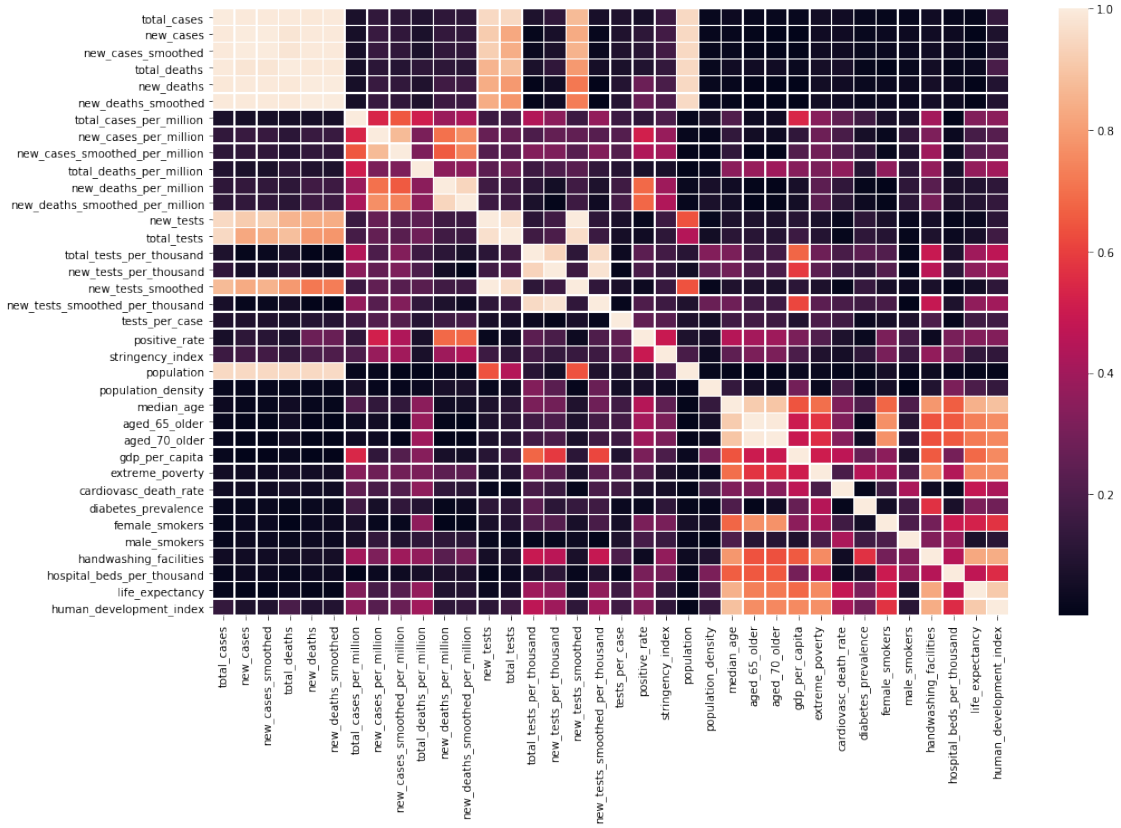
```
46714          0.091  tests performed            80.560   14862927.000
46990            nan              NaN               nan 7794798729.000
47266            nan              NaN               nan            nan

       population_density  median_age  aged_65_older  aged_70_older  \
136                584.800      41.200         13.085          7.452
412                 54.422      18.600          2.581          1.337
606                 23.890      16.800          2.405          1.362
795                    nan         nan            nan            nan
1002               104.871      38.000         13.188          8.643
…                    …          …              …              …
46322               46.754      27.300          5.344          3.053
46519               22.995      17.700          2.480          1.542
46714               42.729      19.600          2.822          1.882
46990               58.045      30.900          8.696          5.355
47266                  nan         nan            nan            nan

       gdp_per_capita  extreme_poverty  cardiovasc_death_rate  \
136         35973.781              nan                    nan
412          1803.987              nan                597.029
606          5819.495              nan                276.045
795               nan              nan                    nan
1002        11803.431            1.100                304.195
…               …                …                      …
46322       12294.876           18.900                200.380
46519        3689.251           57.500                234.499
46714        1899.775           21.400                307.846
46990       15469.207           10.000                233.070
47266             nan              nan                    nan

       diabetes_prevalence  female_smokers  male_smokers  \
136                 11.620             nan           nan
412                  9.590             nan           nan
606                  3.940             nan           nan
795                    nan             nan           nan
1002                10.080           7.100        51.200
…                     …               …             …
46322                5.520           8.100        33.200
46519                3.940           3.100        24.700
46714                1.820           1.600        30.700
46990                8.510           6.434        34.635
47266                  nan             nan           nan

       handwashing_facilities  hospital_beds_per_thousand  life_expectancy  \
136                       nan                         nan           76.290
412                    37.746                       0.500           64.830
606                    26.664                         nan           61.150
```

```
795                        nan                          nan           81.880
1002                       nan                        2.890           78.570
…                          …                            …               …
46322                   43.993                        2.320           64.130
46519                   13.938                        2.000           63.890
46714                   36.791                        1.700           61.490
46990                   60.130                        2.705           72.580
47266                      nan                          nan              nan

       human_development_index   datetime
136                        nan 2020-08-01
412                      0.498 2020-08-01
606                      0.581 2020-08-01
795                        nan 2020-08-01
1002                     0.785 2020-08-01
…                        …        …
46322                    0.699 2020-08-01
46519                    0.588 2020-08-01
46714                    0.535 2020-08-01
46990                      nan 2020-08-01
47266                      nan 2020-08-01

[211 rows x 42 columns]
```

[13]:
```
f, ax = plt.subplots(figsize=(16, 10))
sns.heatmap(covid_agost.corr(), annot=False, fmt="f", linewidths=.5, ax=ax)
f, ax = plt.subplots(figsize=(16, 10))
sns.heatmap(abs(covid_agost.corr()), annot=False, fmt="f", linewidths=.5, ax=ax)
```

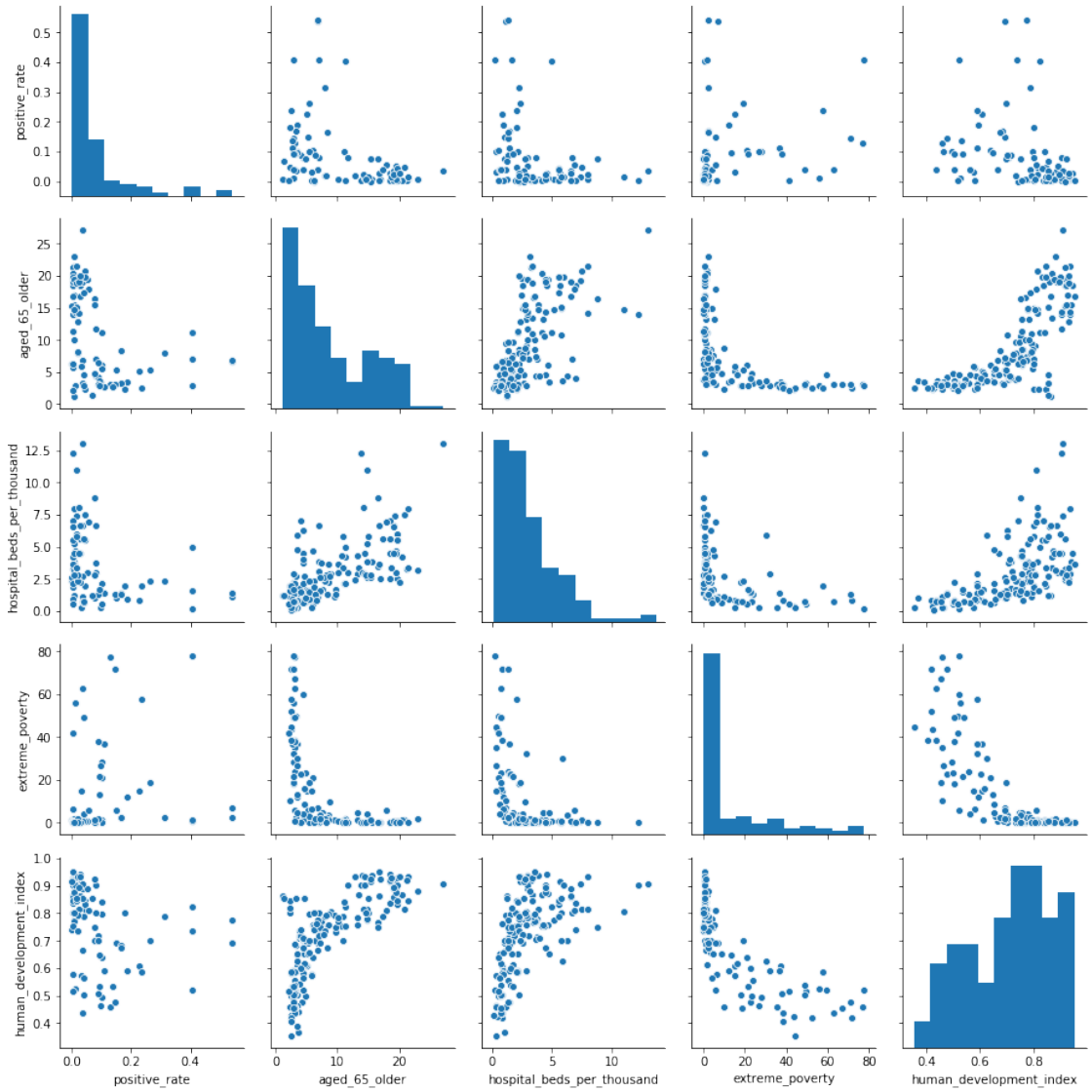
[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdf9dfb1690>
```


total_cases
new_cases
new_cases_smoothed
total_deaths
new_deaths
new_deaths_smoothed
total_cases_per_million
new_cases_per_million
new_cases_smoothed_per_million
total_deaths_per_million
new_deaths_per_million
new_deaths_smoothed_per_million
new_tests
total_tests
total_tests_per_thousand
new_tests_per_thousand
new_tests_smoothed
new_tests_smoothed_per_thousand
tests_per_case
positive_rate
stringency_index
population
population_density
median_age
aged_65_older
aged_70_older
gdp_per_capita
extreme_poverty
cardiovasc_death_rate
diabetes_prevalence
female_smokers
male_smokers
handwashing_facilities
hospital_beds_per_thousand
life_expectancy
human_development_index
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



[14]:
```
selected_columns = ["positive_rate", "aged_65_older",␣
↪"hospital_beds_per_thousand", "extreme_poverty", "human_development_index"]
sns.pairplot(covid_agost[selected_columns])
```

[14]: <seaborn.axisgrid.PairGrid at 0x7fdf9dd6e850>



A continuació, teniu unes funcions que de ben segur haurieu de completar per tal de poder respondre a les preguntes. Podeu implementar-les vosaltres mateixos i llavors comparar-les amb funcions de sklearn (si existeixen)

[15]:
```
from sklearn.model_selection import train_test_split




def separate_dataset(data,target):
  return  train_test_split(
      data[:,[x for x in range(len(data[0])) if x!=target]], data[:,target],␣
 ↪test_size=0.33, random_state=42)
```

```
def to_qualy_mat(data,target):
  qual_quant= {  k: "Qualitatativa " if data[k].dtype  ==object  else␣
 ↪"Quantitativa" for k in data if k != 'datetime' and k!='Timestamp'}
  X = data[[x for x in qual_quant if qual_quant[x]=='Quantitativa']]
  target_index = findIndexColumns(X,target)
  return separate_dataset(X.values,target_index),X


def mostra_correlacions(data):
    f, ax = plt.subplots(figsize=(16, 10))
    return sns.heatmap(data.corr(), annot=False, fmt="f", linewidths=.5, ax=ax)

def normalitzador_de_dades(X,target=None):
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler.fit(X)
    return scaler.transform(X )

def normalizer2(X,target=None):
  from sklearn.preprocessing import MinMaxScaler
  scaler = MinMaxScaler((-1,1))
  scaler.fit(X)
  return scaler.transform(X),scaler

def separar_train_test(data, target, ratio=0.8):
    # TODO retorna les dades en dos subconjunts diferents amb un ratio de␣
 ↪'ratio'
    index = int(data.shape[0]*ratio)
    data_train = data[:index]
    data_val = data[index:]
    data_train = data_train[:,[x for x in range(len(data_train[0])) if x!
 ↪=target]]
    data_val = data_val[:,[x for x in range(len(data_val[0])) if x!=target]]
    target_train = data[:index,target]
    target_val = data[index:,target]
    return data_train, data_val, target_train, target_val

def findIndexColumns(dataset,target):
    return [x for y,x in zip(dataset,range(len(dataset))) if y==target][0]


def aplicar_regressor_lineal(data, target):
    prediccions = []
    # TODO aplica un regressor lineal amb les dades per predir el target
```

```
    return prediccions

from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
def calcula_metrica_error(prediccions, target, tipus):
    # TODO utilitza varies metriques segons la var "tipus" que analitzin les␣
 ↪prediccions
    if tipus == "mse":
        return mean_squared_error(target, prediccions)
    elif tipus == "r2":
        return r2_score(target, prediccions)
    else:
        print ("Metrica {} no reconeguda".format(tipus))
        return -1
```

1.3 A. Exploració de la base de dades (30%)

En aquest apartat aprendrem a analitzar les dades de la base de dades. Les següents funcions ens poden anar bé:

- pandas.read_csv
- DataFrame.groupby
- DataFrame.corr
- sns.pairplot

Podeu veure algunes idees de https://www.kaggle.com/therealcyberlord/coronavirus-covid-19-visualization-prediction

Descripció dels camps: https://github.com/owid/covid-19-data/blob/master/public/data/owid-covid-codebook.csv

1.3.1 Preguntes a respondre

1. Dimensionalitat de la BBDD. Quants exemples, quantes caracteristiques tenim.
2. Com són les caracteristiques?
3. Hi tenim totes les dades (quin % de dades tenim)
4. Quin tipus de atributs tenim a la base de dades.
5. Mostra els atributs més rellevants.
6. Quins atributs están més correlacionats. Mostra'ls.
7. Mostra (almenys) 5 tipus diferents de gràfiques sobre les dades.
8. Els valors es troben tots a la mateixa escala? Ens importa?
9. Quins atributs tenen una distribució Guassiana?
10. Quin és l'atribut objectiu? Per què?

1.4 1.- Dimensionalitat de la BBDD. Quants exemples, quantes caracteristiques tenim

La dimensionalitat es mostra en format nombre de : (nombre de mosters, nombre característiques)

[16]:
```
import datetime
sns.set(rc={'figure.figsize':(11.7,8.27)})
print(f'La bbd del covid té una dimensionaliat de {dataset.shape}')
```

```
La bbd del covid té una dimensionaliat de (47328, 42)
```

1.5 2.- Com són les caracteristiques?

A continuació es mostra cada atribut definit per una clase de python i un nombre de NaN per atribut.

[17]:
```
df = dataset.isna()
print("\t Es mostren en format tipus de variable , nombre de Nans")
df2 = pd.DataFrame({k:(type(dataset[k][0]),sum(df[k])) for k in dataset})
df2['type'] = ['class','NaN number x Class']
df2.set_index('type')
```

```
        Es mostren en format tipus de variable , nombre de Nans
```

[17]:

type	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	total_cases_per_million	new_cases_per_million	new_cases_smoothed_per_million
class	<class 'str'>	<class 'str'>	<class 'str'>	<class 'str'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>	<class 'numpy.float64'>
NaN number x Class	276	552	0	0	614	826	1608	614	826	1608	890	890	1673

```
                  total_deaths_per_million   new_deaths_per_million  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                      890                      890

                  new_deaths_smoothed_per_million                new_tests  \
type
class                       <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                             1673                    30306

                               total_tests total_tests_per_thousand  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                    29898                    29898

                    new_tests_per_thousand       new_tests_smoothed  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                    30306                    28132

                  new_tests_smoothed_per_thousand           tests_per_case  \
type
class                       <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                             28132                    29663

                             positive_rate      tests_units  \
type
class               <class 'numpy.float64'>  <class 'float'>
NaN number x Class                    29230            27297

                            stringency_index               population  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                     7964                      276

                         population_density               median_age  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                     2427                     5133

                              aged_65_older            aged_70_older  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                     5761                     5352

                               gdp_per_capita          extreme_poverty  \
```

```
type
class                <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                      5679                    19538

                       cardiovasc_death_rate      diabetes_prevalence  \
type
class                <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                      5125                     3635

                              female_smokers             male_smokers  \
type
class                <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                     14274                    14693

                       handwashing_facilities hospital_beds_per_thousand  \
type
class                <class 'numpy.float64'>    <class 'numpy.float64'>
NaN number x Class                     27562                       9208

                              life_expectancy  human_development_index  \
type
class                <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                       870                     6611

                                                              datetime
type
class                <class 'pandas._libs.tslibs.timestamps.Timesta…
NaN number x Class                                                 0
```

1.6 3.- Hi tenim totes les dades (quin % de dades tenim)

[18]:
```
df = dataset.isna()
df3 = (pd.DataFrame({k:[(1- df[k].sum()/len(dataset[k]))*100] for k in df}))
df3['index'] = ['tant x 100']
df3.set_index('index')
```

[18]:
```
            iso_code  continent  location    date  total_cases  new_cases  \
index
tant x 100    99.417     98.834   100.000 100.000       98.703     98.255

            new_cases_smoothed  total_deaths  new_deaths  new_deaths_smoothed  \
index
tant x 100              96.602        98.703      98.255               96.602

            total_cases_per_million  new_cases_per_million  \
index
tant x 100                   98.120                 98.120
```

```
            new_cases_smoothed_per_million  total_deaths_per_million  \
index
tant x 100                          96.465                    98.120

            new_deaths_per_million  new_deaths_smoothed_per_million  \
index
tant x 100                  98.120                           96.465

            new_tests  total_tests  total_tests_per_thousand  \
index
tant x 100     35.966       36.828                    36.828

            new_tests_per_thousand  new_tests_smoothed  \
index
tant x 100                  35.966              40.559

            new_tests_smoothed_per_thousand  tests_per_case  positive_rate  \
index
tant x 100                           40.559          37.325         38.240

            tests_units  stringency_index  population  population_density  \
index
tant x 100       42.324            83.173      99.417              94.872

            median_age  aged_65_older  aged_70_older  gdp_per_capita  \
index
tant x 100      89.154         87.828         88.692          88.001

            extreme_poverty  cardiovasc_death_rate  diabetes_prevalence  \
index
tant x 100           58.718                 89.171               92.320

            female_smokers  male_smokers  handwashing_facilities  \
index
tant x 100          69.840        68.955                  41.764

            hospital_beds_per_thousand  life_expectancy  \
index
tant x 100                      80.544           98.162

            human_development_index  datetime
index
tant x 100                   86.032   100.000
```

1.7 4.- Quin tipus de atributs tenim a la base de dades.

Tal i com anteriorment s'ha mostrat els tipus de variable per cada atribut, ara es classifiquen en atribut qualitatiu o quantitatiu; això es fa degut a que ens serà molt útil a l'hora de fer la regressió.

[19]:
```
qual_quant= {  k: ["Qualitatativa " if type(dataset[k][0]) in␣
 ↪(str,type(datetime.datetime.now()))  else "Quantitativa"] for k in dataset}
df4 = pd.DataFrame(qual_quant)
df4['index'] = ['QUAL / QUANT']
df4.set_index('index')
```

[19]:

index	iso_code	continent	location	date
QUAL / QUANT	Qualitatativa	Qualitatativa	Qualitatativa	Qualitatativa

index	total_cases	new_cases	new_cases_smoothed	total_deaths
QUAL / QUANT	Quantitativa	Quantitativa	Quantitativa	Quantitativa

index	new_deaths	new_deaths_smoothed	total_cases_per_million
QUAL / QUANT	Quantitativa	Quantitativa	Quantitativa

index	new_cases_per_million	new_cases_smoothed_per_million
QUAL / QUANT	Quantitativa	Quantitativa

index	total_deaths_per_million	new_deaths_per_million
QUAL / QUANT	Quantitativa	Quantitativa

index	new_deaths_smoothed_per_million	new_tests	total_tests
QUAL / QUANT	Quantitativa	Quantitativa	Quantitativa

index	total_tests_per_thousand	new_tests_per_thousand
QUAL / QUANT	Quantitativa	Quantitativa

index	new_tests_smoothed	new_tests_smoothed_per_thousand
QUAL / QUANT	Quantitativa	Quantitativa

index	tests_per_case	positive_rate	tests_units	stringency_index
QUAL / QUANT	Quantitativa	Quantitativa	Quantitativa	Quantitativa

index	population	population_density	median_age	aged_65_older

```
index
QUAL / QUANT  Quantitativa       Quantitativa  Quantitativa  Quantitativa

              aged_70_older gdp_per_capita extreme_poverty  \
index
QUAL / QUANT  Quantitativa   Quantitativa    Quantitativa

              cardiovasc_death_rate diabetes_prevalence female_smokers  \
index
QUAL / QUANT          Quantitativa        Quantitativa   Quantitativa

               male_smokers handwashing_facilities hospital_beds_per_thousand  \
index
QUAL / QUANT  Quantitativa           Quantitativa               Quantitativa

              life_expectancy human_development_index      datetime
index
QUAL / QUANT    Quantitativa            Quantitativa  Quantitativa
```

1.8 5.- Mostra els atributs mes rellevants

Els atributs més importants i dels quals se'n por extreure mes informació quantitatva son:

[20]:
```
most_imp_cols =␣
 ↪['new_cases_smoothed','new_cases_smoothed',"new_deaths_smoothed","new_deaths_smoothed",'new
dataset[most_imp_cols]
```

[20]:
```
       new_cases_smoothed  new_cases_smoothed  new_deaths_smoothed  \
0                     nan                 nan                  nan
1                   0.286               0.286                0.000
2                   0.286               0.286                0.000
3                   0.286               0.286                0.000
4                   0.286               0.286                0.000
…                   …                   …                    …
47323                 nan                 nan                  nan
47324                 nan                 nan                  nan
47325                 nan                 nan                  nan
47326                 nan                 nan                  nan
47327                 nan                 nan                  nan

       new_deaths_smoothed  new_tests  tests_per_case  positive_rate  \
0                      nan        nan             nan            nan
1                    0.000        nan             nan            nan
2                    0.000        nan             nan            nan
3                    0.000        nan             nan            nan
4                    0.000        nan             nan            nan
…                    …          …               …              …
```

```
47323                  nan        nan             nan             nan
47324                  nan        nan             nan             nan
47325                  nan        nan             nan             nan
47326                  nan        nan             nan             nan
47327                  nan        nan             nan             nan

       population_density  cardiovasc_death_rate  human_development_index
0                 584.800                    nan                      nan
1                 584.800                    nan                      nan
2                 584.800                    nan                      nan
3                 584.800                    nan                      nan
4                 584.800                    nan                      nan
…                     …                      …                        …
47323                 nan                    nan                      nan
47324                 nan                    nan                      nan
47325                 nan                    nan                      nan
47326                 nan                    nan                      nan
47327                 nan                    nan                      nan

[47328 rows x 10 columns]
```

1.9 6.- Quins atributs están més correlacionats. Mostra'ls.

[21]:
```
corr = dataset.corr()
#Eliminem outliers que no donen informació destacable
similitud_threshold = 0.97
corr[corr>similitud_threshold]  = 0

#S'aagafen de dos en dos perque estan mutuament relacionat i les parelles es␣
 ↪repeteixen en ordre
pos_corr = sorted([(x,y,corr[x][y]) for x in corr for y in corr],key = lambda x:
 ↪ x[2],reverse=True)[:20:2]
temp = {'ATT1':[],'ATT2':[],'CORR':[]}
for x in pos_corr:
  temp['ATT1'].append(x[0])
  temp['ATT2'].append(x[1])
  temp['CORR'].append(x[2])
pd.DataFrame(temp)
```

[21]:
```
                    ATT1                             ATT2  CORR
0      new_cases_smoothed                     total_deaths 0.969
1              new_cases                     total_deaths 0.961
2            total_cases              new_cases_smoothed 0.956
3            total_cases                       new_cases 0.946
4            total_cases                     total_tests 0.936
5  new_tests_per_thousand  new_tests_smoothed_per_thousand 0.936
```

```
6            total_tests              new_tests_smoothed 0.921
7         life_expectancy          human_development_index 0.914
8             median_age                    aged_65_older 0.912
9            total_cases                        new_tests 0.910
```

[22]:
```
neg_corr = sorted([(x,y,corr[x][y]) for x in corr for y in corr],key = lambda x:
 ↪ x[2],reverse=False)[:20:2]
temp = {'ATT1':[],'ATT2':[],'CORR':[]}
for x in neg_corr:
  temp['ATT1'].append(x[0])
  temp['ATT2'].append(x[1])
  temp['CORR'].append(x[2])
pd.DataFrame(temp)
```

[22]:

	ATT1	ATT2	CORR
0	extreme_poverty	handwashing_facilities	-0.768
1	extreme_poverty	human_development_index	-0.768
2	extreme_poverty	life_expectancy	-0.750
3	median_age	extreme_poverty	-0.693
4	aged_65_older	extreme_poverty	-0.568
5	aged_70_older	extreme_poverty	-0.550
6	cardiovasc_death_rate	life_expectancy	-0.497
7	gdp_per_capita	extreme_poverty	-0.496
8	gdp_per_capita	cardiovasc_death_rate	-0.481
9	cardiovasc_death_rate	human_development_index	-0.443

1.10 7.- Mostra (almenys) 5 tipus diferents de gràfiques sobre les dades.

En aquesta grafica es preten ensenyar com s'ha augmentat la quanitat de tests a fer com a mesura de prevenció davant del gran nombre de morts que hi habia. Es dona informació de com ha sigut a Europa i de com ha pasat a EEUU.

Europa

[23]:
```
from plotly.offline import init_notebook_mode, iplot
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
init_notebook_mode(connected=True)
fig = go.Figure()
data1 = dataset[dataset.continent=='Europe']
fig = px.line(data1, x="datetime", y="new_tests", color='location')
fig.show()

fig = px.line(data1, x="datetime", y="new_deaths", color='location')
fig.show()
```

EEUU

[24]:
```
fig = go.Figure()
data1 = dataset[dataset.continent=='North America']
fig = px.line(data1, x="datetime", y="new_tests", color='location')
fig.show()

fig = px.line(data1, x="datetime", y="new_deaths", color='location')
fig.show()
```

Es curiós com les morts en EEUU van començçar a creixer desmesuradament fins que es van començçar a fer testos de manera progressiva creixent a la població fins que s'ha estabilitzat el nombre de morts així com la quantitat de nous testos.

Es sorprenent també la manca de resposta per part de méxic essent el 2n país mes afectat.

Impacte global de les morts Es interessant que es pot eliminar llocs del mon clicant a la llegenda, el mes interessant es clicar a "World" a la llegenda per veure les proporcions reals.

[25]:
```
specs = [[{'type':'domain'}, {'type':'domain'}]]

fig = go.Figure()
data1 = dataset.groupby([dataset.location]).sum().reset_index()
fig = px.pie(dataset, values='total_deaths', names='location',
             title='Deaths over the world',
             hover_data=['total_deaths'], labels={'lifeExp':'life expectancy'})
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

Si treiem el mon del grafic (prement-lo en la llegenda); podem apreciar que EEUU ha rebut mes d un cinqué de les morts a nivell mundial i que juntament amb Brazil i México fa que el continent Americà sigui el mes afectat per mortalitat de virus amb més del **40% de les morts mundials**.

Aportacions dels països a la quantitat de casos

[26]:
```
df2 = dataset[dataset.location!='World']
fig=px.bar(
    data_frame = df2,
    x = df2['datetime'],
    y = df2['new_cases'],
    color = 'location',
    color_discrete_sequence = px.colors.qualitative.Light24,


)
fig.show()
```

D'aquesta gràfica es pot apreciar com va ser China el primer i únic punt zero.

Podem dir també que els països que es poden arribar a trobar amb una situació tant dolenta com la que hem vist als EEUU son Argentina i India que tenen una gran taxa de nous infectats creixent.

Grafica concienciació Amb aquesta gràfica es vol fer veure que totes les conclusions tretes fins ara també s'han de pendre amb discrecció degut a que el nivell de desenvolupament d'un país esta directament relacionat amb les probes que es fan i, per tant ,amb els casos que es detecten. (Podria ser que EEUU sigui qui tingui mes casos per tindre millor index de desenvolupament i fer mes testos per contindre d'una manera mes eficient el virus)

De fet països que viuen en la miseria o que directament es neguen a fer pública aquesta informació poden estar amagant una gran quantitat de casos i morts

[27]:
```
g= sns.relplot(data=dataset,
               x="new_tests",
               y="total_cases",
               hue='human_development_index',
               size='human_development_index',
               sizes=(50,200),
               palette='gist_heat',
               height=12,

              )

g.fig.suptitle("Money Means Tests")
```

[27]:
```
Text(0.5, 0.98, 'Money Means Tests')
```

Grafic distribució Gaussiana En aquest grafic es vol mostrar els atributs que estan distribuïts de manera mes propera a una normal $N\ (\mu, \sigma)$ sent μ la mitjana de la distribució i σ la desviació estàndard de la mateixa. Despres es seleccionen les 7 millors i es mostren les distribucions i les seves gaussianes associades.

Com a conclusió ningún atribut s'assembla suficient a una normal per poder dir que la segueix tot i que en algun podem veure rasgos de normal.

[28]:
```
#Quins atributs tenen una distribució Guassiana?
data = dataset
mu, sigma = 0, 0.2 # media y desvio estandar
datos = np.random.normal(mu, sigma, 1000) #creando muestra de datos
qual_quant= {  k: "Qualitatativa " if data[k].dtype  ==object  else␣
 ↪"Quantitativa" for k in data if k != 'datetime'}
X = data[[x for x in qual_quant if qual_quant[x]=='Quantitativa']]
errors = []
info2=  {}
for x in X:
```

```
    gauss= np.random.normal(X[x].mean(), X[x].std(), len(X[x]))
    df =pd.DataFrame(dict(
      series=np.concatenate(([x]*len(X[x]), ["gauss"]*len(gauss))),
      data  =np.concatenate((X[x],gauss))
  ))

    errors.append((((X[x]-gauss)**2).sum(),x))

    info2[x] = df

sorterr = sorted(errors,key=lambda x:x[0])
N=7
for x in sorterr[:N]:
  fig = px.histogram(info2[x[1]], x="data", color="series",␣
 ↪barmode="overlay",title=x[1]+'+ Gaussian('+x[1]+')')
  fig.show()
```

Fent agregacions d'atributs per tal de trobar representacions temporals de la població ens podrien portar a distribucions mes semblants a una normal; tot i axi podem concluïr que ningun atribut de per si té una distribució semblant a una normal de manera natural.

1.11 8.- Els valors es troben tots a la mateixa escala? Ens importa?

No, els valors no es troben en la mateixa escala. Podem dir que en principi si el metode que es fa servir es el dels mínims quadrats (scikit.LinearRegressor) no influirà tant(tot i que influirà) en els resultats. Per altra banda si fem servir l'algoritme de descens del gradient a diferents escales estem donant-li directament mes pes a aquestes variables en comptes de decidir l'algoritme del descens quin es el pes que li pertoca. **A tot això tenint en compte que aplicarem normalització, no ens importa.**

1.12 9.- Quins atributs tenen una distribució Guassiana?

S'ha aprofitat en la pregunta 7 per ficar un tipus de grafics que son de distribució de les dades(últims grafics) . Els atrtibuts escollits son aquells que s'assemblen mes a una gaussiana amb mitja i desviació del atribut. Com a funció de costat s'ha agafat la distancia L_2 entre les dues distribucions. S'han agafat les 7 mes properes i s'ha mostrat.

Com a conclusió afegir que de cap de les distibucions s'hi assembla de maner apura , tot i que alguna combinacions d'atributs agrupades temporalment s'hi haurien d'assemblar,

1.13 10.- Quin és l'atribut objectiu? Per què?

D'atributs objectius en podem considerar dos de molt obvis. Aquest atributs son **new_deaths i new_cases**. El primer es vol predir per intentar ser minimitzat i l'altre es molt important sobre tot a nivell logístic i económic. Degut a la manca d'interés economic en aquest treball ens centrarem en el factor humà; es a dir **el nostre atribut objectiu serà new_deaths**

1.14 B. Ús de regressor Lineal (30%)

Funcions a tenir en compte

- data preprocessing
- sklearn.pipeline.make_pipeline
- sklearn.linear_model.LinearRegression
- sklearn.linear_model.SGDRegressor
- sklearn.metrics.mean_squared_error

1.14.1 Preguntes a respondre

1. Aprén un Regressor Lineal amb totes les dades
2. Calcula l'error quadràtic mitjà del regressor per a cada un dels atributs de la base de dades. Quin atribut té l'error més baix. Mostra'ls.
3. És millor o pitjor que utilitzant totes les dades? Per què?
4. Tenen alguna relació els atributs amb distribucions Gaussianes i els que tenen un error més petit?
5. Què passa si normalitzes les dades? El error és més baix?
6. Significa això que el regressor és més precís? Passa el mateix amb altres mètriques?
7. Heu après un LinearRegression o un SGDRegressor? Sabeu quines diferencies hi ha? Compareu-los Pista

1.15 1.- Aprén un Regressor Lineal amb totes les dades

[29]:
```
from sklearn.model_selection import train_test_split

(data_train, data_val, target_train, target_val), X =␣
↪to_qualy_mat(dataset,"new_deaths")

from sklearn.linear_model import LinearRegression
#DATA T AND DATA V ES FAN SERVIR PER ENTRENAR
dataTrain = [(x,y) for x,y in zip(data_train,target_train) if (not np.any(pd.
↪isna(x)) and not np.any(pd.isna(y)))]
dataT = np.asarray([x[0] for x in dataTrain])
dataV = np.asarray([x[1] for x in dataTrain])
#DATA VT AND DATA V ES FAN SERVIR PER MIRAR RESULTATS
dataVal = np.asarray([(x,y) for x,y in zip(data_val,target_val) if (not np.
↪any(pd.isna(x)) and not np.any(pd.isna(y)))])
dataVT = np.asarray([x[0] for x in dataVal])
dataVV = np.asarray([x[1] for x in dataVal])

modelRegression = LinearRegression().fit(dataT,dataV)
mse = calcula_metrica_error(modelRegression.predict(dataVT) , dataVV,'mse')
r2 = calcula_metrica_error(modelRegression.predict(dataVT) , dataVV,'r2')
#total_prediction_error = np.mean((modelRegression.predict(dataVT) - dataVV)**2)
```

```
print(f"Per a un model amb totes les dades el MSE obitngut  es de {mse}")
print(f"Per a un model amb totes les dades el r2 obitngut  es de {r2}")
```

```
Per a un model amb totes les dades el MSE obitngut  es de 1106.6952920168942
Per a un model amb totes les dades el r2 obitngut  es de 0.9614927948351563
```

1.16 2.- Calcula l'error quadràtic mitjà del regressor per a cada un dels atributs de la base de dades. Quin atribut té l'error més baix. Mostra'ls.

[30]:
```
import time
info = {}
(data_train, data_val, target_train, target_val), X =␣
 ↪to_qualy_mat(dataset,"new_deaths")
X = X[[x for x in X if x !=  "new_deaths"]]
for column in X:
  if 'new_deaths' not in column:
    variable = findIndexColumns(X,column)
    dataTrain = [(x,y) for x,y in zip(data_train[:,variable],target_train) if␣
 ↪(not np.any(pd.isna(x)) and not np.any(pd.isna(y)))]
    dataT = np.asarray([x[0] for x in dataTrain])
    dataV = np.asarray([x[1] for x in dataTrain])
    dataVal = np.asarray([(x,y) for x,y in zip(data_val[:,variable],target_val)␣
 ↪if (not np.any(pd.isna(x)) and not np.any(pd.isna(y)))])
    dataVT = np.asarray([x[0] for x in dataVal])
    dataVV = np.asarray([x[1] for x in dataVal])
    start = time.time()
    modelRegression = LinearRegression().fit(dataT.reshape(-1,1),dataV)
    mse = calcula_metrica_error(modelRegression.predict(dataVT.reshape(-1,1)) ␣
 ↪, dataVV,'mse')
    r2 = calcula_metrica_error(modelRegression.predict(dataVT.reshape(-1,1))  ,␣
 ↪dataVV,'r2')
    info[column] = [time.time()-start]
    info[column].append(mse)
    info[column].append(r2)
indexes = sorted(info,key = lambda x : info[x][2],reverse=True)
values = [info[x][1] for x in indexes]
times  = [info[x][0] for x in indexes]
r2 = [info[x][2] for x in indexes]
print("Sorted by r2 Values:")
pd.DataFrame({'Lowest MSE' : indexes,'MSE': values,'r2':r2,'Time':times})
```

```
Sorted by r2 Values:
```

[30]:

	Lowest MSE	MSE	r2	Time
0	new_cases	30828.428	0.785	0.001
1	new_cases_smoothed	34803.906	0.763	0.001
2	total_deaths	42816.089	0.702	0.001

```
3                       total_cases  59046.749  0.588 0.001
4                        population  60188.695  0.581 0.001
..                                ...        ...    ...   ...
27                     aged_70_older 160440.777  0.000 0.001
28                   life_expectancy 145490.419  0.000 0.001
29                    gdp_per_capita 161767.930 -0.000 0.001
30  new_tests_smoothed_per_thousand  28071.454 -0.000 0.001
31         total_tests_per_thousand  26542.718 -0.001 0.001

[32 rows x 4 columns]
```

[31]:
```
print('Sorted by MSE values : ')
indexes = sorted(info,key = lambda x : info[x][1],reverse=False)
values = [info[x][1] for x in indexes]
times  = [info[x][0] for x in indexes]
r2 = [info[x][2] for x in indexes]
pd.DataFrame({'Lowest MSE' : indexes,'MSE': values,'r2':r2,'Time':times})
```

```
Sorted by MSE values :
```

[31]:
```
                     Lowest MSE        MSE    r2  Time
0                     new_tests  14785.416 0.444 0.001
1       human_development_index  16619.511 0.010 0.001
2              stringency_index  16678.405 0.014 0.001
3            new_tests_smoothed  17487.965 0.377 0.001
4                   total_tests  18750.387 0.293 0.001
..                          ...        ...   ...   ...
27  hospital_beds_per_thousand 176706.505 0.001 0.001
28              female_smokers 203067.826 0.000 0.001
29                male_smokers 204152.627 0.000 0.001
30             extreme_poverty 238629.235 0.001 0.001
31      handwashing_facilities 308652.925 0.001 0.001

[32 rows x 4 columns]
```

La metrica important a tindre en compte es r2 perque , en resum, ens diu com de predible es el target en funció dels inputs

WIKIPEDIA DEFINEIX R2 COM: "___ the proportion of variance in the dependent variable that is predictable from the independent variable(s)"

1.17 3.- És millor o pitjor que utilitzant totes les dades? Per què?

Es molt pitjor que fent servir totes les dades(sempre i quan hi hagi regularitzador) , això es deu a que la funció que mapeja cap a les noves morts no té una única variable com a input (si fos tan facil ja hauriem solucionat el problema). A mes ames si alguna variable no esta gens relacionada amb el resultat esperat el model ja s'encarrega "d'eliminar-la" si es necesari.

Sense regularitzador res assegura que s'eliminin les menys importants(en cas del model SGD o GD).

Basant-nos en la metrica r2 el millor resultat possible per a variables individuals es 0.785 mentre que por totes les variables es 0.96

1.18 4.- Tenen alguna relació els atributs amb distribucions Gaussianes i els que tenen un error més petit?

Seguint el ultim grafic de la pregunta 7 del apartat A

No hi ha una relació evidenciable pero si que es cert que alguns dels atributs com human_dvelopment_index o stringency_index estan adalt en la classificaicó de millors atributs segons MSE, tot i que no es el cas per la metrica R2 que es la que ens interessa

1.19 5.-Què passa si normalitzes les dades? El error és més baix?

[32]:
```
(data_train, data_val, target_train, target_val), X =␣
↪to_qualy_mat(dataset,"new_deaths")

dataTrain = [(x,y) for x,y in zip(data_train,target_train) if (not np.any(pd.
↪isna(x)) and not np.any(pd.isna(y)))]
dataT = np.asarray([x[0] for x in dataTrain])
dataV = normalitzador_de_dades(np.asarray([x[1] for x in dataTrain]).
↪reshape(-1,1))
dataVal = np.asarray([(x,y) for x,y in zip(data_val,target_val) if (not np.
↪any(pd.isna(x)) and not np.any(pd.isna(y)))])
dataVT = np.asarray([x[0] for x in dataVal])
dataVV = normalitzador_de_dades(np.asarray([x[1] for x in dataVal]).
↪reshape(-1,1))

#NOrmalitzem tot el que no es predicció degut a que deconeixem els estadistics␣
↪de les prediccions
dataT = normalitzador_de_dades(dataT)
dataVT  = normalitzador_de_dades(dataVT)


modelRegression = LinearRegression().fit(dataT,dataV)
mse = calcula_metrica_error(modelRegression.predict(dataVT)  , dataVV,'mse')
r2 = calcula_metrica_error(modelRegression.predict(dataVT) , dataVV,'r2')
normalized_error = np.mean((modelRegression.predict(dataVT) - dataVV)**2)
print(f"Mantenint les dades normalitzades el MSE error es : {mse}")
print(f"Mantenint les dades normalitzades el R2 error es : {r2}")
```

```
Mantenint les dades normalitzades el MSE error es : 0.0360251923657213
Mantenint les dades normalitzades el R2 error es : 0.9639748076342787
```

En normalitzar totes les dades veiem que el MSE baixa drasticament, això es deu a que aquesta metrica es directament dependent de la escala dels outputs, si normalitzem els outputs el MSE es reduirà drasticament.

Tot i així la metrica que ens interessa que es independent de la escala de les dades(R2) segueix

mantenint-se com s'esperaba.

1.20 6.- Significa això que el regressor és més precís? Passa el mateix amb altres mètriques?

No, el regressor no es mes precis; la escala del output depén de la del input i la esclaa del MSE depen de la escala del output. Així doncs un input amb una gran escala farà que el MSE sigui molt alt. Mentre que un input normalitzat farà que el MSE es mantingui molt baix.

Com s'ha dit la metrica r2, de la que ens fiem, es manté "igual" normalitzant que sense normalitzar.

1.21 7.- Heu après un LinearRegression o un SGDRegressor? Sabeu quines diferencies hi ha? Compareu-los

Fins ara hem apres un model LinearRegression. La diferencia principal entre ambdós son els algoritmes que fan servir per resoldre el problema. El regresor lineal resol un problema de mínims quadrats de manera analítica (probablemennt fent servir el algoritme OLS (ordinary least squares) extés a mes variables). Mentre que el SGDRegressor realment implementa el algoritme iteratiu de descens del gradient (Stochastic Gradiant Descence) que nosaltres implementarem més endevant.

Tot i així el nostre algoritme implementat no es el SGD; s'aplica un gradient descent(GD) directament. El algoritme de scikit afegeixen components estocastics o heurístics en els que no s'entrarà.

1.22 C. Demostració d'assoliment (A+B en una nova BBDD) (20%)

La puntuació d'aquesta secció dependrà de la originalitat, i el treball realizat l'analisis i procesat d'una base de dades alternativa de lliure elecció. Recordeu que podeu reaprofitar la majoria del codi si ho heu implementat en funcions.

Es seguirà la mateixa metodologia que en el apartat A per tant només s'especifiacrà les diferencies substancials.

1.22.1 C.A.1.- Dimensionalitat de la BBDD. Quants exemples, quantes caracteristiques tenim

[33]:
```
df_p1 = pd.read_csv("Plant_1_Generation_Data.csv")
df_p2 = pd.read_csv("Plant_2_Generation_Data.csv")
df_w1 = pd.read_csv("Plant_1_Weather_Sensor_Data.csv")
df_w2 = pd.read_csv("Plant_2_Weather_Sensor_Data.csv")
print(f"Ens trobem davant de una database que preten enfrontar la eficiencia de␣
 ↪dues plantes eléctriques; per a fer-ho de cada planta tenim dos tipus de␣
 ↪característiques:",
      f"\n\tPlanta 1",
          f"\n\t\t Característiques generació elèctrica: {df_p1.shape}",
          f"\n\t\t Característiques climatiques {df_w1.shape}",
          f"\n\t\t Nombre d'inverters: {df_p1['SOURCE_KEY'].nunique()}",
      f"\n\tPlanta 2",
          f"\n\t\t Característiques generació elèctrica: {df_p2.shape}",
          f"\n\t\t Característiques climatiques {df_w2.shape}",
        f"\n\t\t Nombre d'inverters: {df_p1['SOURCE_KEY'].nunique()}")
```

```
print(f"Observem que no tenen exactament el mateix tamany, en un futur proper␣
 ↪s'haura d'arreglar aquest problema per facilitar els calculs vectorials")
```

```
Ens trobem davant de una database que preten enfrontar la eficiencia de dues
plantes eléctriques; per a fer-ho de cada planta tenim dos tipus de
característiques:
        Planta 1
                Característiques generació elèctrica: (68778, 7)
                Característiques climatiques (3182, 6)
                Nombre d'inverters: 22
        Planta 2
                Característiques generació elèctrica: (67698, 7)
                Característiques climatiques (3259, 6)
                Nombre d'inverters: 22
Observem que no tenen exactament el mateix tamany, en un futur proper s'haura
d'arreglar aquest problema per facilitar els calculs vectorials
```

1.22.2 C.A.2.- Com són les caracteristiques?

[34]:
```
def caracterize(dataset):
  df = dataset.isna()
  print("\t Es mostren en format tipus de variable , nombre de Nans")
  df2 = pd.DataFrame({k:(type(dataset[k][0]),sum(df[k])) for k in dataset})
  df2['type'] = ['class','NaN number x Class']

  return df2.set_index('type')
```

Planta 1

[35]:
```
caracterize(df_p1)
```

```
        Es mostren en format tipus de variable , nombre de Nans
```

[35]:

type	DATE_TIME	PLANT_ID	SOURCE_KEY
class	<class 'str'>	<class 'numpy.int64'>	<class 'str'>
NaN number x Class	0	0	0

type	DC_POWER	AC_POWER
class	<class 'numpy.float64'>	<class 'numpy.float64'>
NaN number x Class	0	0

type	DAILY_YIELD	TOTAL_YIELD
class	<class 'numpy.float64'>	<class 'numpy.float64'>
NaN number x Class	0	0

[36]:
```
caracterize(df_w1)
```

```
        Es mostren en format tipus de variable , nombre de Nans
```

[36]:
```
                        DATE_TIME              PLANT_ID     SOURCE_KEY  \
type
class               <class 'str'>  <class 'numpy.int64'>  <class 'str'>
NaN number x Class              0                     0              0

                        AMBIENT_TEMPERATURE       MODULE_TEMPERATURE  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                        0                        0

                                  IRRADIATION
type
class               <class 'numpy.float64'>
NaN number x Class                        0
```

Planta 2

[37]:
```
caracterize(df_p2)
```

```
        Es mostren en format tipus de variable , nombre de Nans
```

[37]:
```
                        DATE_TIME              PLANT_ID     SOURCE_KEY  \
type
class               <class 'str'>  <class 'numpy.int64'>  <class 'str'>
NaN number x Class              0                     0              0

                                     DC_POWER                 AC_POWER  \
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                        0                        0

                                  DAILY_YIELD              TOTAL_YIELD
type
class               <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                        0                        0
```

[38]:
```
caracterize(df_w2)
```

```
        Es mostren en format tipus de variable , nombre de Nans
```

[38]:
```
                        DATE_TIME              PLANT_ID     SOURCE_KEY  \
type
class               <class 'str'>  <class 'numpy.int64'>  <class 'str'>
NaN number x Class              0                     0              0
```

```
                         AMBIENT_TEMPERATURE       MODULE_TEMPERATURE  \
type
class                <class 'numpy.float64'>  <class 'numpy.float64'>
NaN number x Class                         0                        0

                                  IRRADIATION
type
class                <class 'numpy.float64'>
NaN number x Class                         0
```

1.22.3 C.A.3-Hi tenim totes les dades? Quin % hi tenim?

Com es pot apreciar a les celes anteriors als datasets no hi ha cap NaN per tant podem concloure que tenim un 100% de les dades.

1.22.4 C.A.4 Quin tipus de atributs tenim a la base de dades.

Es pot veure en la pregunta C.A.2 que per ambdues plantes les dades de generació electrica tenen **data de registre, identificador de planta i identificador de sensor que interve** com a atributs qualitatius i dades sobre **la potencia de corrent (alter i continu) i el total acumulat generat** com a dades quantitatives.

Pel que fa a les dades ambientals també son iguals per ambdués plantes; té els mateixos identificadors qualitatius que el dataset de generació electrica peró les mesures quantitatives son diferents , aquí es mesura **temperatura ambiental, temperatura del modul i irradiació**

1.22.5 C.A. 5- Mostra els atributs mes rellevants

Info Plantes

[39]:
```
print(f"Els atributs mes rellevants son els id dels inverters, i les corrents␣
 ↪que generaben en funcio de la hora i les condicions climàtiques")
df_p1['DATE_TIME'] = pd.to_datetime(df_p1['DATE_TIME'],format ='%d-%m-%Y %H:%M')
df_p2['DATE_TIME'] = pd.to_datetime(df_p2['DATE_TIME'],format ='%Y-%m-%d %H:%M:
 ↪%S')
df_p1['date'] = df_p1['DATE_TIME'].dt.date
df_p1['time'] = df_p1['DATE_TIME'].dt.time
df_p2['date'] = df_p2['DATE_TIME'].dt.date
df_p2['time'] = df_p2['DATE_TIME'].dt.time
df_w1['DATE_TIME'] = pd.to_datetime(df_w1['DATE_TIME'],format ='%Y-%m-%d %H:%M:
 ↪%S')
df_w2['DATE_TIME'] = pd.to_datetime(df_w2['DATE_TIME'],format ='%Y-%m-%d %H:%M:
 ↪%S')
df_w1['date'] = df_w1['DATE_TIME'].dt.date
df_w1['time'] = df_w1['DATE_TIME'].dt.time
df_w2['date'] = df_w2['DATE_TIME'].dt.date
df_w2['time'] = df_w2['DATE_TIME'].dt.time
```

```
grouped = df_p1.copy().append(df_p2)
grouped = grouped.groupby([grouped.PLANT_ID,grouped.SOURCE_KEY]).mean()
print("Dades importans per planta i per inverter")
grouped
```

```
Els atributs mes rellevants son els id dels inverters, i les corrents que
generaben en funcio de la hora i les condicions climàtiques
Dades importans per planta i per inverter
```

[39]:

		PLANT_ID	DC_POWER	AC_POWER	DAILY_YIELD	\
PLANT_ID	SOURCE_KEY					
4135001	1BY6WEcLGh8j5v7	4135001	2873.721	281.124	3230.658	
	1IF53ai7Xc0U56Y	4135001	3226.337	315.488	3356.882	
	3PZuoBAID5Wc2HD	4135001	3221.016	314.967	3356.856	
	7JYdWkrLSPkdwr4	4135001	3132.892	306.386	3325.810	
	McdE0feGgRqW7Ca	4135001	3211.307	314.046	3321.936	
…		…	…	…	…	
4136001	q49J1IKaHRwDQnt	4136001	226.162	221.252	3751.280	
	rrq4fwE8jgrTyWY	4136001	209.158	204.665	2680.059	
	vOuJvMaM2sgwLmb	4136001	262.097	256.217	3513.997	
	xMbIugepa2P7lBB	4136001	280.712	274.505	4901.645	
	xoJJ8DcxJEcupym	4136001	240.927	235.634	3282.404	

		TOTAL_YIELD
PLANT_ID	SOURCE_KEY	
4135001	1BY6WEcLGh8j5v7	6377463.855
	1IF53ai7Xc0U56Y	6313086.323
	3PZuoBAID5Wc2HD	7117012.541
	7JYdWkrLSPkdwr4	7728914.419
	McdE0feGgRqW7Ca	7287579.907
…		…
4136001	q49J1IKaHRwDQnt	430333.957
	rrq4fwE8jgrTyWY	120176223.990
	vOuJvMaM2sgwLmb	2291490.056
	xMbIugepa2P7lBB	105820299.148
	xoJJ8DcxJEcupym	207719486.756

```
[44 rows x 5 columns]
```

Info Plantes + Weather La informacio amb la que es pot extreure mes coneixement es:

[40]:
```
source_key1 = df_p1[df_p1['SOURCE_KEY']=='1BY6WEcLGh8j5v7'].reset_index()
source_key_info = source_key1.
 ↪merge(df_w1,left_on=['date','time'],right_on=['date','time'])
source =␣
 ↪source_key_info[['date','time','SOURCE_KEY_x','SOURCE_KEY_y','DC_POWER','AC_POWER','DAILY_Y
 ↪copy()
```

```
source
```

[40]:

	date	time	SOURCE_KEY_x	SOURCE_KEY_y	DC_POWER	\
0	2020-05-15	00:00:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
1	2020-05-15	00:15:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
2	2020-05-15	00:30:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
3	2020-05-15	00:45:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
4	2020-05-15	01:00:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
…	…	…	…	…	…	
3149	2020-06-17	22:45:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
3150	2020-06-17	23:00:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
3151	2020-06-17	23:15:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
3152	2020-06-17	23:30:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	
3153	2020-06-17	23:45:00	1BY6WEcLGh8j5v7	HmiyD2TTLFNqkNe	0.000	

	AC_POWER	DAILY_YIELD	TOTAL_YIELD	AMBIENT_TEMPERATURE	\
0	0.000	0.000	6259559.000	25.184	
1	0.000	0.000	6259559.000	25.085	
2	0.000	0.000	6259559.000	24.936	
3	0.000	0.000	6259559.000	24.846	
4	0.000	0.000	6259559.000	24.622	
…	…	…	…	…	
3149	0.000	5521.000	6485319.000	22.151	
3150	0.000	5521.000	6485319.000	22.130	
3151	0.000	5521.000	6485319.000	22.008	
3152	0.000	5521.000	6485319.000	21.969	
3153	0.000	5521.000	6485319.000	21.909	

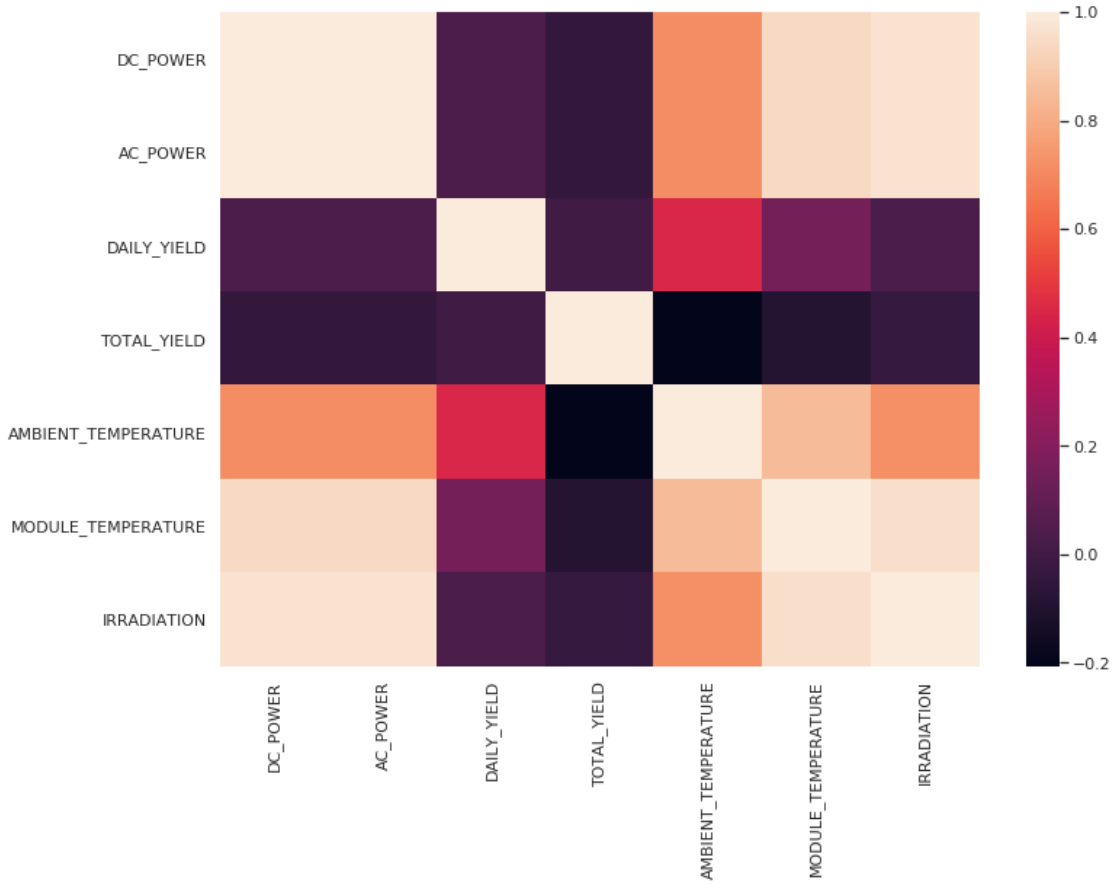
	MODULE_TEMPERATURE	IRRADIATION
0	22.858	0.000
1	22.762	0.000
2	22.592	0.000
3	22.361	0.000
4	22.165	0.000
…	…	…
3149	21.480	0.000
3150	21.389	0.000
3151	20.709	0.000
3152	20.735	0.000
3153	20.428	0.000

[3154 rows x 11 columns]

1.22.6 C.A.6.- Quins atributs están més correlacionats. Mostra'ls.

[41]:
```
sns.heatmap(source.corr())
print(f"Es pot apreciar que hi ha una forta relació entre tots els atributs del␣
 ↪conjunt [ DC_POWER,␣
 ↪AC_POWER,AMBIENT_TEMPERATURE,MODULE_TEMPERATURE,IRRADIATION ] ")
print(f"S aprecia també com per si soles aquestes característiques no son␣
 ↪capaces de correlacionar-se fortament amb els nostres targets: els valors de␣
 ↪corrent mantingut")
```

```
Es pot apreciar que hi ha una forta relació entre tots els atributs del conjunt
[ DC_POWER, AC_POWER,AMBIENT_TEMPERATURE,MODULE_TEMPERATURE,IRRADIATION ]
S aprecia també com per si soles aquestes característiques no son capaces de
correlacionar-se fortament amb els nostres targets: els valors de corrent
mantingut
```



1.22.7 C.A.7.- Mostra (almenys) 5 tipus diferents de gràfiques sobre les dades.

[42]:
```
import plotly.graph_objects as go
```

DC POWER per planta

[43]:
```
data1= df_p1
data2 = df_p2

fig = go.Figure()

fig.add_trace(go.Scattergl(x=data1['time'],
                           y=data1['DC_POWER'],
                           mode='markers',
                           marker=dict(
                               size=4,
                               color= data1['DC_POWER'],
                               cauto=True,
                               colorscale ='Oryel',
                               opacity=0.3
                           ),
                           name='Plant 1 DC power'))

fig.add_trace(go.Scatter(x=data1['time'],
                           y=data1.groupby('time').mean()['DC_POWER'],
                           mode='lines',
                               line=dict(
                               color='DarkGray',
                               width=3
                           ),
                           name='Plant 1 Mean'))

fig.add_trace(go.Scattergl(x=data2['time'],
                           y=data2['DC_POWER'],
                           mode='markers',
                           marker=dict(
                               size=4,
                               color= data2['DC_POWER'],
                               cauto=True,
                               colorscale ='Blugrn',
                               opacity=0.3
                           ),
                           name='Plant 2 DC power'))

fig.add_trace(go.Scatter(x=data2['time'],
                           y=data2.groupby('time').mean()['DC_POWER'],
                           mode='lines',
                               line=dict(
```

```
                            color='DarkOliveGreen',
                            width=3
                        ),
                        name='Plant 2 Mean'))

fig.update_layout(title= 'DC Power Generation by time',
                  height = 600)
fig.show()
```

La conclusió clara es que en les hores de incisió màxima del sol sobre les plantes la planta nombre 1 es 7 cops mes eficient que la planta nombre 2.

AC POWER per inverter over TIME

[44]:
```
import plotly.express as px

data=df_p1.groupby(['SOURCE_KEY','date']).sum().reset_index()

data['source_key'] = data['SOURCE_KEY']
data2=df_p2.groupby(['SOURCE_KEY','date']).sum().reset_index()

data2['source_key'] = data2['SOURCE_KEY']
fig=px.bar(
    data_frame = data,
    x = data['date'],
    y = data['AC_POWER'],
    color = 'source_key',
    color_discrete_sequence = px.colors.qualitative.Light24,
    title='Plant 1 AC POWER Distribution'

)


fig.show()


fig=px.bar(
    data_frame = data2,
    x = data2['date'],
    y = data2['AC_POWER'],
    color = 'source_key',
    color_discrete_sequence = px.colors.qualitative.Light24,
        title='Plant 2 AC POWER Distribution'


)
```

```
fig.show()
```

Com a informació a extreure sembla que les aportacions de la planta 1 estan millor repartides mentre que la càrreca de treball dels inverters a la planta 2 està molt desequilibrada. Aquesta mateixa te el pic mes alt d'ambdues grafiques a començament d'any però no hi torna a aconseguir la mateixa producció, mentre que la 1 repartint millor la fina assoleix el pic de 600k varies vegades.

Relacio Temperatura ambient amb Temperatura modul

[45]:
```
full_data = df_p1.merge(df_w1,left_on=['date','time'],right_on=['date','time'])
full_data2 = df_p2.merge(df_w2,left_on=['date','time'],right_on=['date','time'])
data = full_data.append(full_data2).reset_index()


data = data.groupby(['PLANT_ID_x','date','time']).mean().reset_index()
data = data.drop(['date','time'], axis=1)
sns.relplot(data=data,
            x="AMBIENT_TEMPERATURE",
            y="MODULE_TEMPERATURE",
            hue='IRRADIATION',
            size='IRRADIATION',
            sizes=(50,200),
            palette='gist_heat',
            height=12,
            col='PLANT_ID_x'
           )
```

[45]:
```
<seaborn.axisgrid.FacetGrid at 0x7fdf70157450>
```

Sembla que el gràfic de la planta 1 mostra que el modul s'escalfa una miqueta mes rapid que no pas en la planta 2, entenent que això millora la producció energetica

Contribució de cada inverter en la generació d'energia de la planta

[46]:
```
from plotly.subplots import make_subplots
data1=df_p1.groupby(['SOURCE_KEY']).sum().reset_index()
data1['SOURCE_KEY'] = data1['SOURCE_KEY']
data2=df_p2.groupby(['SOURCE_KEY']).sum().reset_index()
data2['SOURCE_KEY'] = data2['SOURCE_KEY']

specs = [[{'type':'domain'}, {'type':'domain'}]]
fig = make_subplots(rows=1, cols=2, specs=specs)

pull_factor = [0]*22
pull_factor[7] = 0.05

fig.add_trace(go.Pie(labels='P1 '+ data1['SOURCE_KEY'],
                     values=data1['AC_POWER'],
                     name='Plant 1',
                     title='Plant 1',
                     titlefont=dict(
                                    size=25
                                   ),
                     hovertemplate="%{label} <br />generates %{value:,.0f} kW",
                     marker_colors = px.colors.qualitative.Dark24,
                     legendgroup = 'Plant 1',
                    ), 1, 1)

fig.add_trace(go.Pie(labels='P2 '+ data2['SOURCE_KEY'],
                     values=data2['AC_POWER'],
                     name='Plant 2',
                     title='Plant 2',
                     titlefont=dict(
                                    size=25
                                   ),
                     hovertemplate="%{label} <br />generates %{value:,.0f} kW",
                     marker_colors = px.colors.qualitative.Light24,
                     legendgroup = 'Plant 2',
                     pull =pull_factor,
                    ), 1, 2)

fig.update_traces(hole=.4)

fig.update_layout(
    title_text="AC Power Generation of each Inverter"
)


fig.show()
```

Atributs i Gaussianes

[47]:
```
data = dataset
mu, sigma = 0, 0.2 # media y desvio estandar
datos = np.random.normal(mu, sigma, 1000) #creando muestra de datos

errors = []
info2=  {}

X =␣
 ↪full_data[['AC_POWER','DC_POWER','DAILY_YIELD','TOTAL_YIELD','AMBIENT_TEMPERATURE','MODULE_
for x in X:
    gauss= np.random.normal(X[x].mean(), X[x].std(), len(X[x]))
    df =pd.DataFrame(dict(
      series=np.concatenate(([x]*len(X[x]), ["gauss"]*len(gauss))),
      data  =np.concatenate((X[x],gauss))
  ))

    errors.append((((X[x]-gauss)**2).sum(),x))

    info2[x] = df

sorterr = sorted(errors,key=lambda x:x[0])
N=7
for x in sorterr[:N]:
  fig = px.histogram(info2[x[1]], x="data", color="series",␣
 ↪barmode="overlay",title=x[1]+'+ Gaussian('+x[1]+')')
  fig.show()
```

Les variables no s'assemblen gaire a una normal tot i que la irradiació i les temperatures sense els outliers s'hi podria arribar a assemblar

1.22.8 C.A.8.- Els valors es troben tots a la mateixa escala? Ens importa?

NO, no tots i la resposta a la importancia esta en A.8

C.A.9 .-Quins atributs tenen una distribució Guassiana? Com s'ha dit en el peu de les grafiques de les gaussianes, les temperatures i la irradiació son les que s'apropen mes a una normal amb els respectius estadistics

C.A.10 - Quin és l'atribut objectiu? Per què? El atribut objectiu serà o TOTAL YIELD o DAILY YIELD, potser seria mes convenient DAILI_YIELD perque aquest genera el TOTAL_YIELD

1.22.9 C.B.1 Aprén un Regressor Lineal amb totes les dades

[48]:
```
F= full_data[[x for x in full_data if 'DATE_TIME' not in x]]
```

[49]:
```
F
```

[49]:
```
       PLANT_ID_x     SOURCE_KEY_x  DC_POWER  AC_POWER  DAILY_YIELD  \
0         4135001  1BY6WEcLGh8j5v7     0.000     0.000        0.000
1         4135001  1IF53ai7Xc0U56Y     0.000     0.000        0.000
2         4135001  3PZuoBAID5Wc2HD     0.000     0.000        0.000
3         4135001  7JYdWkrLSPkdwr4     0.000     0.000        0.000
4         4135001  McdE0feGgRqW7Ca     0.000     0.000        0.000
…              …                …         …         …            …
68769     4135001  uHbuxQJl8lW7ozc     0.000     0.000     5967.000
68770     4135001  wCURE6d3bPkepu2     0.000     0.000     5147.625
68771     4135001  z9Y9gH1T5YWrNuG     0.000     0.000     5819.000
68772     4135001  zBIq5rxdHJRwDNY     0.000     0.000     5817.000
68773     4135001  zVJPv84UY57bAof     0.000     0.000     5910.000

       TOTAL_YIELD        date      time  PLANT_ID_y     SOURCE_KEY_y  \
0      6259559.000  2020-05-15  00:00:00     4135001  HmiyD2TTLFNqkNe
1      6183645.000  2020-05-15  00:00:00     4135001  HmiyD2TTLFNqkNe
2      6987759.000  2020-05-15  00:00:00     4135001  HmiyD2TTLFNqkNe
3      7602960.000  2020-05-15  00:00:00     4135001  HmiyD2TTLFNqkNe
4      7158964.000  2020-05-15  00:00:00     4135001  HmiyD2TTLFNqkNe
…              …          …         …           …                …
68769  7287002.000  2020-06-17  23:45:00     4135001  HmiyD2TTLFNqkNe
68770  7028601.000  2020-06-17  23:45:00     4135001  HmiyD2TTLFNqkNe
68771  7251204.000  2020-06-17  23:45:00     4135001  HmiyD2TTLFNqkNe
68772  6583369.000  2020-06-17  23:45:00     4135001  HmiyD2TTLFNqkNe
68773  7363272.000  2020-06-17  23:45:00     4135001  HmiyD2TTLFNqkNe

       AMBIENT_TEMPERATURE  MODULE_TEMPERATURE  IRRADIATION
0                   25.184              22.858        0.000
1                   25.184              22.858        0.000
2                   25.184              22.858        0.000
3                   25.184              22.858        0.000
4                   25.184              22.858        0.000
…                        …                   …            …
68769               21.909              20.428        0.000
68770               21.909              20.428        0.000
68771               21.909              20.428        0.000
68772               21.909              20.428        0.000
68773               21.909              20.428        0.000

[68774 rows x 13 columns]
```

[50]:
```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression

(data_train, data_val, target_train, target_val),X =␣
 ↪to_qualy_mat(F,'DAILY_YIELD')
modelRegression = LinearRegression().fit(data_train,target_train)
mse = calcula_metrica_error(modelRegression.predict(data_val) ,␣
 ↪target_val,'mse')
r2 = calcula_metrica_error(modelRegression.predict(data_val) , target_val,'r2')

print(f"Per a un model amb totes les dades el MSE obtingut es de {mse}")
print(f"Per a un model amb totes les dades el R2 obtingut es de {r2}")
```

```
Per a un model amb totes les dades el MSE obtingut es de 5964788.657980355
Per a un model amb totes les dades el R2 obtingut es de 0.3977583795050068
```

1.22.10 C.B.2.- Calcula l'error quadràtic mitjà del regressor per a cada un dels atributs de la base de dades. Quin atribut té l'error més baix. Mostra'ls.

[52]:
```
import time
info = {}

(data_train, data_val, target_train, target_val),X =␣
 ↪to_qualy_mat(F,'DAILY_YIELD')
X = X[[x for x in X if x not in ( "new_deaths","PLANT_ID_x","PLANT_ID_y")]]
for column in X:
  if 'DAILY_YIELD' not in column:
    variable = findIndexColumns(X,column)
    start = time.time()
    modelRegression = LinearRegression().fit(data_train[:,variable].
 ↪reshape(-1,1),target_train)

    mse = calcula_metrica_error(modelRegression.predict(data_val[:,variable].
 ↪reshape(-1,1)) , target_val,'mse')
    r2 = calcula_metrica_error(modelRegression.predict(data_val[:,variable].
 ↪reshape(-1,1))  , target_val,'r2')
    info[column] = [time.time()-start]
    info[column].append(mse)
    info[column].append(r2)
indexes = sorted(info,key = lambda x : info[x][1])
values = [info[x][1] for x in indexes]
times  = [info[x][0] for x in indexes]
r2 = [info[x][2] for x in indexes]
pd.DataFrame({'Lowest MSE' : indexes,'MSE': values,'R2':r2,'Time':times})
```

[52]:
```
             Lowest MSE         MSE     R2  Time
0   MODULE_TEMPERATURE 7648322.747  0.228 0.001
1          IRRADIATION 9524202.589  0.038 0.001
2             AC_POWER 9843914.409  0.006 0.001
3          TOTAL_YIELD 9904127.712  0.000 0.001
4             DC_POWER 9904926.545 -0.000 0.002
5  AMBIENT_TEMPERATURE 9904926.545 -0.000 0.001
```

1.22.11 C.B.3.- És millor o pitjor que utilitzant totes les dades? Per què?

Es pitjor degut a la explicació donada en B.3

1.22.12 C.B.4.- Tenen alguna relació els atributs amb distribucions Gaussianes i els que tenen un error més petit?

Doncs en aquest cas dos de les més semblants a una normal generen el error més petit mentre que la tercera genrea el error mes gran. (Irradiació i temperatura de modul mes relacionades) i la temperatura d'ambient com la menys relacionada.

1.22.13 C.B.5.- Què passa si normalitzes les dades? El error és més baix?

[53]:
```
(data_train, data_val, target_train, target_val),X =␣
↪to_qualy_mat(F,'DAILY_YIELD')

#Normalitzem tot el que no es predicció degut a que deconeixem els estadistics␣
↪de les prediccions
dataT = normalitzador_de_dades(data_train)
dataVT  = normalitzador_de_dades(data_val)
modelRegression = LinearRegression().fit(dataT,target_train)
mse = calcula_metrica_error(modelRegression.predict(dataVT) , target_val,'mse')
r2 = calcula_metrica_error(modelRegression.predict(dataVT)  , target_val,'r2')
print(f"Mantenint les dades normalitzades el MSE es : {mse}")
print(f"Mantenint les dades normalitzades el R2 es : {r2}")
```

```
Mantenint les dades normalitzades el MSE es : 5965914.338963861
Mantenint les dades normalitzades el R2 es : 0.3976447238537275
```

En aquest cas i degut a la poca relacio lineal que hi ha entre les variables obtenim errrors molt als de MSE i una proporcio de variances molt baixa per el R2

1.22.14 C.B.6- Conlusio:

Tot i que es un dataset que demostra no poder predir-se amb un sistema linal està be veure com de baixes son les metriques per poder-nos adonar; veurem si amb el regresor polinomial aconseguim alguna cosa millor.

1.23 D. Implementació Regressor Lineal (20%)

En aquest exercici, es tracta d'implementar en python el procés de descens del gradient explicat a les classes de teoria, i comparar-lo amb els resultats obtinguts amb l'apartat (B).

$$J(w) = \frac{1}{2m}\left[\sum_{i=1}^{m}(f(x^i;w) - y^i)^2 + \lambda\sum_{j=1}^{n}(w_j^2)\right]$$

Fixeu-vos que J retorna el `mse`. Per a trobar w_j, repetir fins convergència:

$$w_0 = w_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(f(x^i;w) - y^i)\cdot 1$$

$$w_j = w_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(f(x^i;w) - y^i)\cdot x_j^i - \frac{\lambda}{m}w_j\right]$$

ó:

$$w_j := w_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{\lambda}{m}\sum_{i=1}^{m}(f(x^i;w) - y^i)\cdot x_j^i$$

On si considerem un regressor lineal (el model és una recta), llavors w_0 i w_1 representen, respectivament, la b i a de la fòrmula de la recta:

$$h_\theta(x^{(i)}) = ax + b$$

α és el learning rate, i $h_\theta(x^{(i)})$ és la funció que fa la regressió, és a dir, la funció que prediu el valor de $y^{(i)}$ donat un(s) atribut(s) concret(s) $x^{(i)}$.

Així, tenint calculat el model en l'últim punt del primer exercici, ja sabeu quin resultat hauríeu d'obtenir. O no, perquè la vostra implementació pot ser millor! En concret, es tracta de desenvolupar aquestes tasques:

- Definir la funció de cost i del gradient
- Estudiar com l'ús de regularitzadors afecta el resultat: overfitting, underfitting, etc.
- Visualització de les dades a analitzar i explicació pas a pas del procediment
- Visualització del procés de descens de gradient
- Modificar el learning rate i el nombre d'iteracions

Per a la implementació us podeu basar en el següent esquelet:

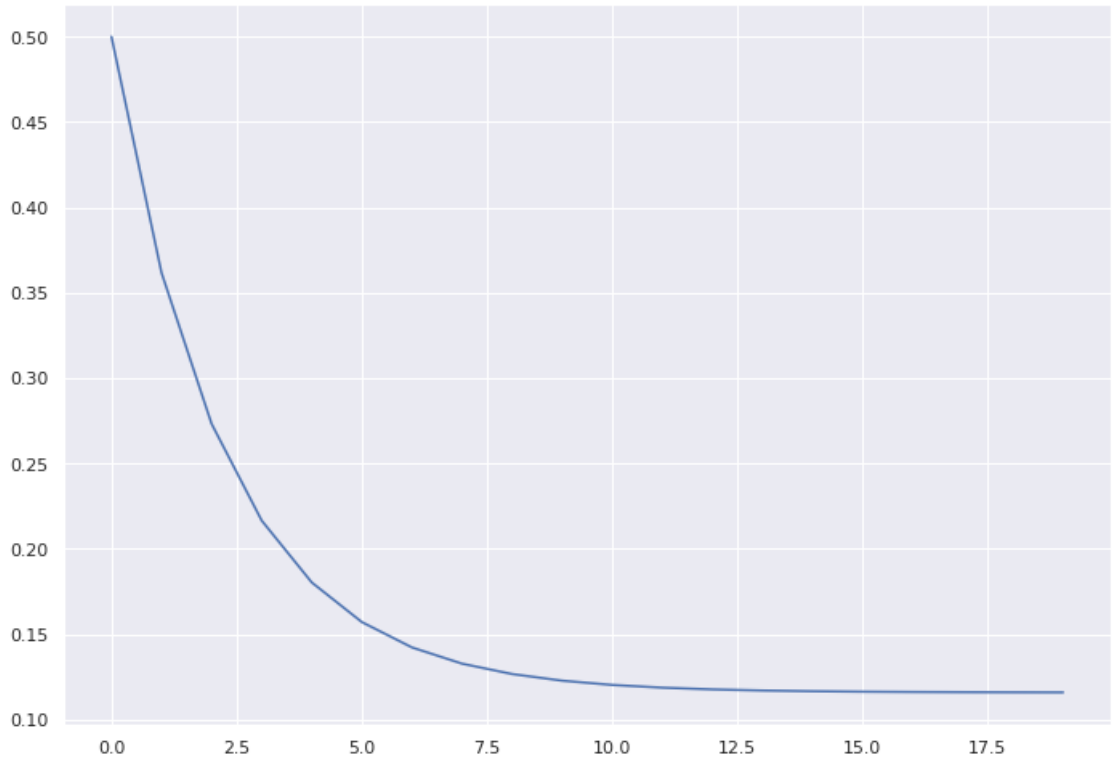
[54]:
```
class Regressor(object):
    def __init__(self, w0, w1, alpha):
        # Inicialitzem w0 i w1
        self.weights= np.asarray([w0,w1],dtype=np.float64).reshape(-1,1)
        self.alpha = alpha
```

```
        self.coefs = []

    def predict(self, x):
        return np.dot(x,self.weights).reshape(1,-1).flatten()

    def __update(self, x, y):

          m=len(x)
          error = self.predict(x) - y
          cost = 1/(2*m) * np.dot(error.T, error)
          self.weights -= (self.alpha * (1/m) * np.dot(x.T, error)).
 ↪reshape(-1,1)
          return cost

    def consecutives_differences(self,array,thr):
       array = np.asarray(array)
       return  np.all(np.abs((array[1:] - array[:-1]))[:-5] > thr)

    def fit(self, x, y, max_iter=1000, epsilon=1e-5,draw=True):
        iter = 0
        error = ([x for x in range(max_iter)])
        while iter < max_iter and self.consecutives_differences(error[:iter if␣
 ↪iter> 5 else 5],epsilon):
            error[iter] = (self.__update(x,y))
            iter+=1
            self.coefs.append(self.weights)
        if draw:sns.lineplot(x=np.arange(iter),y=error[:iter])

        return error



(data_train, data_val, target_train, target_val),X =␣
 ↪to_qualy_mat(dataset,"new_deaths")
variable = findIndexColumns(X,'new_cases')
```

[55]:
```
from ipywidgets import interact_manual,FloatSlider
data = [(x,y) for x,y in zip(data_train[:,variable],target_train) if (not pd.
 ↪isna(x) and not pd.isna(y))]
dataT = np.asarray([x[0] for x in data])
dataV = np.asarray([x[1] for x in data])
x= normalitzador_de_dades(dataT.reshape(-1,1))
y = normalitzador_de_dades(np.asarray([x[1] for x in data]).reshape(-1,1)).
 ↪reshape(1,-1).flatten()
x = np.c_[np.ones(x.shape[0]), x]
def make_regression(w0,w1,alpha,max_iter=1000):
  reg = Regressor(w0,w1,alpha)
```

```
  error = reg.fit(x,y,max_iter=max_iter)
  return reg,error

#reg = interact_manual(make_regression,w0=FloatSlider(min=0, max=1, step=0.
 ↪01),w1=FloatSlider(min=0, max=1, step=0.01),alpha=FloatSlider(min=0.001,␣
 ↪max=1, step=0.01),lambd=FloatSlider(min=0.001, max=1, step=0.01));
min(make_regression(0,0,0.2,20)[1])
```

[55]: 0.11582880323321872



Així es podrà contestar a aquestes preguntes:

1. Com influeixen tots els paràmetres en el procés de descens?
2. Quins valors de learning rate convergeixen més ràpid a la solució òptima?
3. Com influeix la inicialització del model en el resultat final?
4. Quina diferència (quantitativa i qualitativa) hi ha entre el vostre regressor i el de la llibreria ?
5. Evalueu mètriques de execució

1.23.1 1.- Com influeixen tots els paràmetres en el procés de descens?

Els parametres que fem servir principalment en aquest model (sense fer servir el regularitzador que es farà servir i s'explicarà en la part extra) son:

1. Funció d'inicialització de pesos: Desde quin punt de la funció es comença el descens
2. Learning rate: Indica com de grans son els passos cap a la direcció del gradient en cada iteració

1.23.2 2.- Quins valors de learning rate convergeixen més ràpid a la solució òptima?

De manera generica a mes gran sigui el learning rate mes rapida es el descens del gradient. El problema es que si la funció es abrupte i te molts mínims locals podem estar donant voltes infinitament per la superficie(o hiperplà) de la funció. Així que el learning rate s'ha d'adaptar a cada model i atributs. En el nostre cas la convergencia es molt rapida amb un learning rate bastant alt (0.1) en aproximadament 40 iteracions. Augmentar el learning rate només fa que la convergencia tardi mes iteracions; no troba un millor mínim global.

1.23.3 3.- Com influeix la inicialització del model en el resultat final?

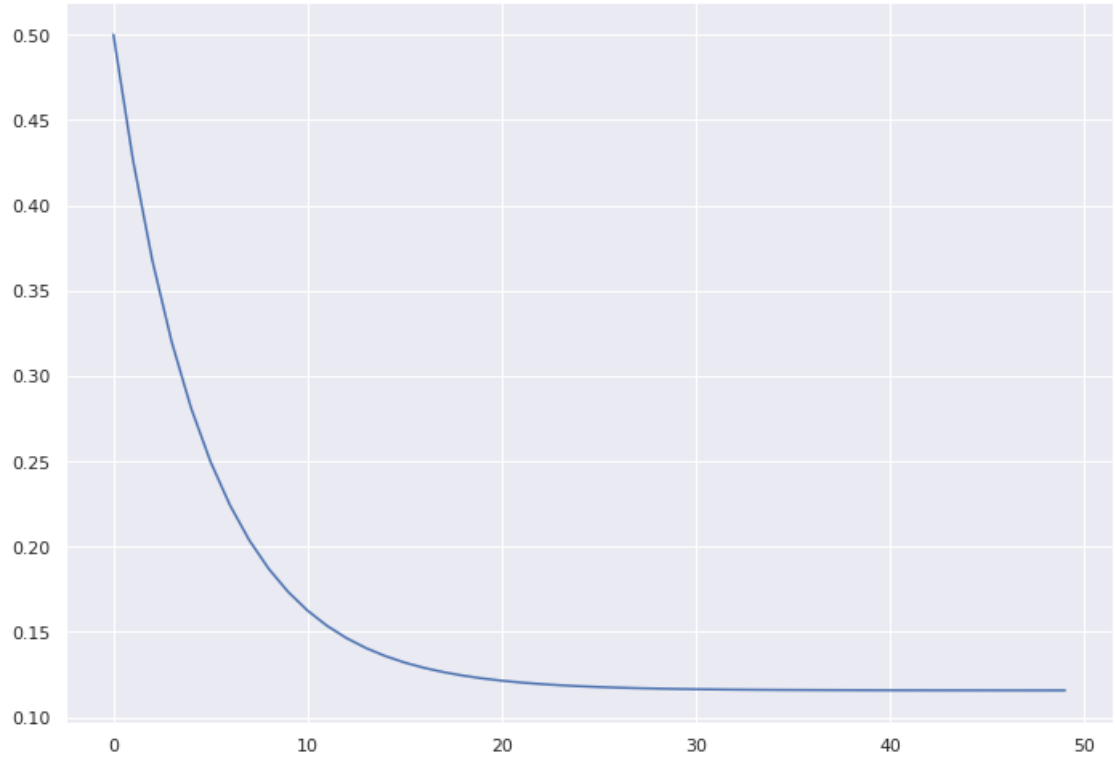
Tal i com s'ha dit; la inicialització del model reflexa en quin punt de la funció comencem del descens. Contra menys mínims locals tingui la funció més indiferent resulta la inicialització. Això es deu a que si la funció es convexa amb un únic mínim global la inicialització es totalment indiferent.

El problema comença quan tenim una funció abrupte amb molts mínims locals que tenen un gran pendent. En aquest cas començar el descens en un punt o altre de la funció significaria trobar o no una solucó óptima.

1.23.4 4.-Quina diferència (quantitativa i qualitativa) hi ha entre el vostre regressor i el de la llibreria ?

[56]:
```
from sklearn.linear_model import SGDRegressor
import random
from ipywidgets import interact_manual,FloatSlider
data = [(x,y) for x,y in zip(data_train[:,variable],target_train) if (not pd.
 ↪isna(x) and not pd.isna(y))]
dataT = np.asarray([x[0] for x in data])
x= normalitzador_de_dades(dataT.reshape(-1,1))
y = normalitzador_de_dades(np.asarray([x[1] for x in data]).reshape(-1,1)).
 ↪reshape(1,-1).flatten()
x = (np.c_[np.ones(x.shape[0]), x] )
#random.shuffle(x)
reg = Regressor(0,0,0.1)
error = reg.fit(x,y,max_iter=1000)


sckreg = SGDRegressor().fit(x,y)
#reg = interact_manual(make_regression,w0=FloatSlider(min=0, max=1, step=0.
 ↪01),w1=FloatSlider(min=0, max=1, step=0.01),alpha=FloatSlider(min=0.001,␣
 ↪max=1, step=0.01),lambd=FloatSlider(min=0.001, max=1, step=0.01));
print(sckreg.coef_)
print(reg.weights.reshape(1,-1))
```

```
[3.96167638e-04 8.85071736e-01]
[[3.67330591e-17 8.72124443e-01]]
```



La diferencia principal que s'aprecia es que el model de la llibreria no es determinista, es a dir en cada fit els pesos varien. Per trobar una estimació inicialitzaré els pesos varies vegades i calcularé la distancia mitjana respecte els del meu model.

[57]:
```
t = np.asarray([SGDRegressor().fit(x,y).coef_ for _ in range(100)])
```

[58]:
```
dist = np.sum((t-reg.weights.flatten()),axis=1)**2
print(f"Mitjana de distancies es : {np.mean(dist)}")
print(f"Maxim de distancies es : {np.max(dist)}")
print(f"Mínim de distancies es : {np.min(dist)}")
```

```
Mitjana de distancies es : 0.003503035537869874
Maxim de distancies es : 0.020739836845378987
Mínim de distancies es : 1.5632898932008373e-08
```

Podem concluir que fa una inicialització aleatoria sense fixar un seed i que fent-ho d'aquesta manera pot acaabr en un uan distancia maxima de coeficients de 0.025

[59]:
```
data = [(x,y) for x,y in zip(data_val[:,variable],target_val) if (not pd.
 ↪isna(x) and not pd.isna(y))]
dataTT = np.asarray([x[0] for x in data])
```

```
dataTV = np.asarray([x[1] for x in data])
normedt = (normalitzador_de_dades(dataTT.reshape(-1,1)))
normedt = np.c_[np.ones(len(normedt)),normedt]
normedv = (normalitzador_de_dades(dataTV.reshape(-1,1)))
error = sckreg.predict(normedt) -normedv.reshape(1,-1).flatten()
total_MSE = np.dot(error.T,error)
error = reg.predict(normedt) - -normedv.reshape(1,-1).flatten()
total_MSE_own = np.dot(error.T,error)
print(f"Libraries model has a validation (Normalized) error of: {total_MSE}")
print(f"Own model has a validation error  of {total_MSE_own}")
```

```
Libraries model has a validation (Normalized) error of: 3292.873315867111
Own model has a validation error  of 50726.39408629747
```

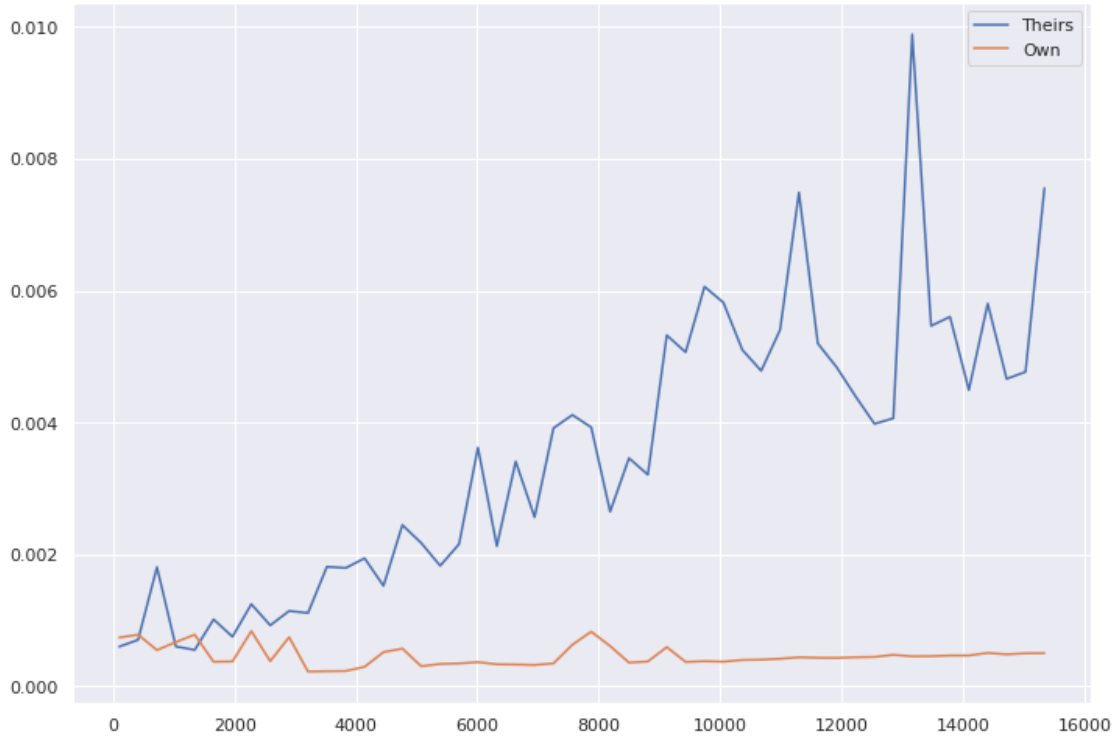
El model de la llibreria desaccaradement te una millor perfromance que el model basic que només aplica l'algoritme de descens del gradient. De fet aquest algoritme fa un descens del gradient barrejat amb un algoritme estocastic que deu millorar la eficiencia del model.

Aquetsa diferencia es dona amb la ordenacio per defecte, si s'aleatoritza la distribució inicial amb un random.shuffle el nostre model aconsegueix una validation error millor. Això fa pensar que la part estocastica del algoritme es distribui-se les dades com millor li convingui i com es estocastic no sempre ho fa igual; per tant es distribueix les dades diferents si desordenem les dades.

###5.- Evalueu mètriques de execució

[60]:
```
import time
X,Y =x,y
info = {'N':[],'TIME':[],'MODEL':[]}
for t in np.linspace(100,len(normedt)):
    info['N'].append(int(t))
    info['MODEL'].append('Theirs')
    time1 = time.time()
    sckreg = SGDRegressor().fit(X[:int(t)],Y[:int(t)])
    info['TIME'].append(time.time()-time1)

    info['N'].append(int(t))
    info['MODEL'].append('Own')
    time1 = time.time()
    reg.fit(X[:int(t)],Y[:int(t)],draw=False)
    info['TIME'].append(time.time()-time1)



sns.lineplot(data=info,x="N",y='TIME',hue='MODEL')
```

[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdf6bf29490>
```



Aquests resultats es deuen a que s'ha cambiat la condicio de parada a quan la diferencia de errors sigui prou petita. En el nostre cas fa que el temps per N=2000 i N= 16000 siguin pracitcament iguals degut a la rapida convergencia

1.24 (Extra) Implementació Regresor polinomial (+10%, max: 10pts)

1. Quins parametres heu de cambiar al vostre Regressor per tal de poder predir amb funcions polinomials?
2. Quines funcions polinomials (de diferent grau, de diferents combinacions d'atributs, ...) heu escollit per ser apreses amb el vostre descens del gradient? quina ha donat el millor resultat (en error i rapidesa en convergència)?
3. Utilitzeu el regularitzador en la fòrmula de funció de cost i descens del gradient i proveu polinomis de diferent grau. Com afecta el valor del regularitzador?
4. Té sentit el model (polinomial) trobat quan es visualitza sobre les dades?

###1.- Quins parametres heu de cambiar al vostre Regressor per tal de poder predir amb funcions polinomials?

Ara en comptes de rebre els pesos explicitament rep un nombre de pesos i una funcio que els inicializta; degut a que abans ja s'ha intentat optimitzar el temps mitjançan operacions vectoritzades amb numpy no s'ha de cambiar l'algoritme

[61]:
```
class RegressorPolynomial(object):
    def __init__(self,number_weights,func_init,alpha,regulizer=None):
      self.weights = np.asarray([func_init(x) for x in range(number_weights+1)])
```

```
      self.alpha = alpha
      self.regulizer = regulizer
    def predict(self,X):
      return X@self.weights

    def __update_regulizing(self, x, y):
        m=len(x)
        error = self.predict(x) - y
        cost = 1/(2*m) * np.dot(error.T, error) + self.regulizer * (np.dot(self.
↪weights.T,self.weights))
        self.weights = self.weights * (1 - (self.alpha * self.regulizer)/m) -␣
↪(self.alpha * (self.regulizer/m) * np.dot(x.T, error)).reshape(1,-1).
↪flatten()
        return cost

    def __update(self, x, y):
        m= len(x)
        error = self.predict(x) - y
        cost = 1/(2*m) * np.dot(error.T, error)
        self.weights -= (self.alpha * (1/m) * np.dot(x.T, error)).reshape(1,-1).
↪flatten()
        return cost

    def consecutives_differences(self,array,thr):

       array = np.asarray(array)
       return  np.all(np.abs((array[1:] - array[:-1]))[:-5] > thr)

    def fit(self, x, y, max_iter=1000,␣
↪epsilon=1e-5,draw=True,untill_convergence=False,reg=False):
        iter = 0
        max_iter = int(1e5) if untill_convergence else max_iter
        error = ([x for x in range(max_iter)])

        gradient_descent = self.__update_regulizing if reg else self.__update

        while iter < max_iter and self.consecutives_differences(error,epsilon):
            error[iter] = gradient_descent(x,y)
            iter+=1
        if draw:sns.lineplot(x=np.arange(iter),y=error[:iter])

        return error

degree = 5
data = [(x,y) for x,y in zip(data_train[:,variable],target_train) if (not pd.
↪isna(x) and not pd.isna(y))]
dataT = np.asarray([[x[0]**y for y in range(degree+1)] for x in data])
```

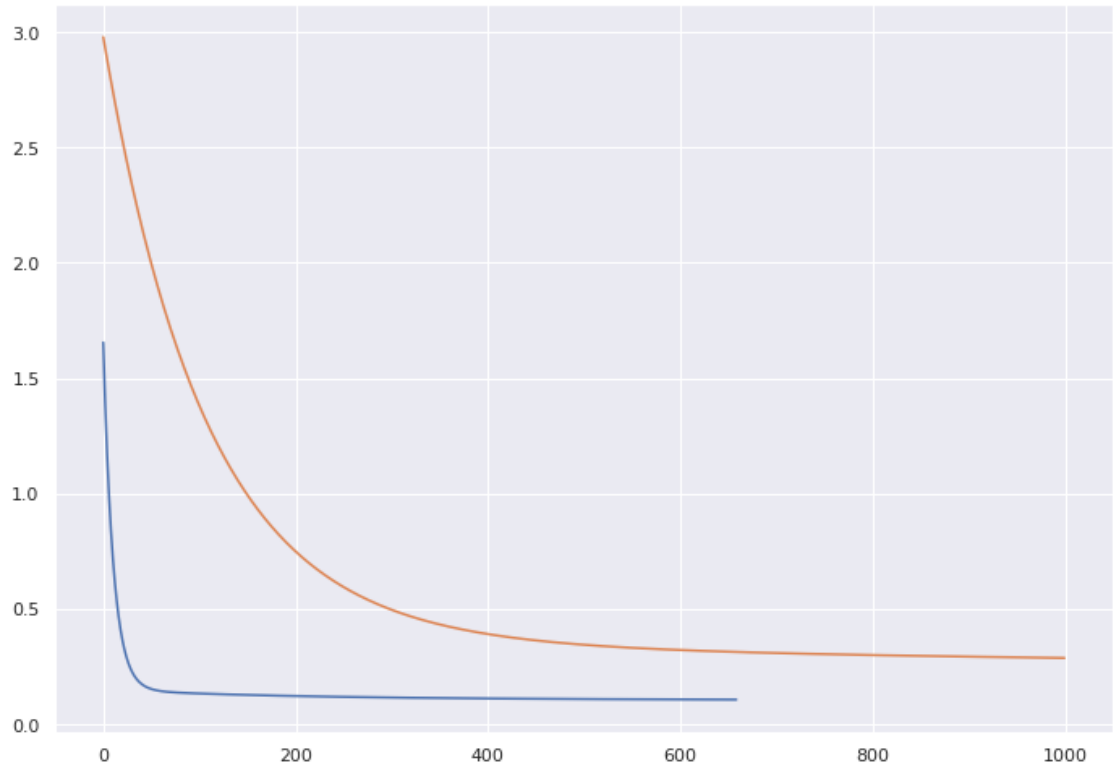
```
dataV = np.asarray([x[1] for x in data])
```

1.24.1 2.- Quines funcions polinomials (de diferent grau, de diferents combinacions d'atributs, ...) heu escollit per ser apreses amb el vostre descens del gradient? quina ha donat el millor resultat (en error i rapidesa en convergència)?

S'ha escollit el atribut casos nous i els seus polinomis de grau N per fer les probes perque s'ha cregut que en algun grau estarien prou propers

1.24.2 3.-Utilitzeu el regularitzador en la fòrmula de funció de cost i descens del gradient i proveu polinomis de diferent grau. Com afecta el valor del regularitzador?

[62]:
```
def make_pol_regression(degree,alpha,lambd,normalitzar = False,regul=False):
  import random
  dataV2 = normalitzador_de_dades(dataV.reshape(-1,1)).reshape(1,-1).flatten()␣
 ↪if normalitzar else dataV
  reg = RegressorPolynomial(degree,lambda x: random.random(),alpha,lambd)
  reg.
 ↪fit(normalitzador_de_dades(dataT),dataV2,untill_convergence=False,reg=regul)
  return reg
#reg = interact_manual(make_regression,w0=FloatSlider(min=0, max=1, step=0.
 ↪01),w1=FloatSlider(min=0, max=1, step=0.01),alpha=FloatSlider(min=0.001,␣
 ↪max=1, step=0.01),lambd=FloatSlider(min=0.001, max=1, step=0.01));
reg = make_pol_regression(5,0.01,0.1,True,False)
reg2 = make_pol_regression(5,0.01,0.1,True,True)
```



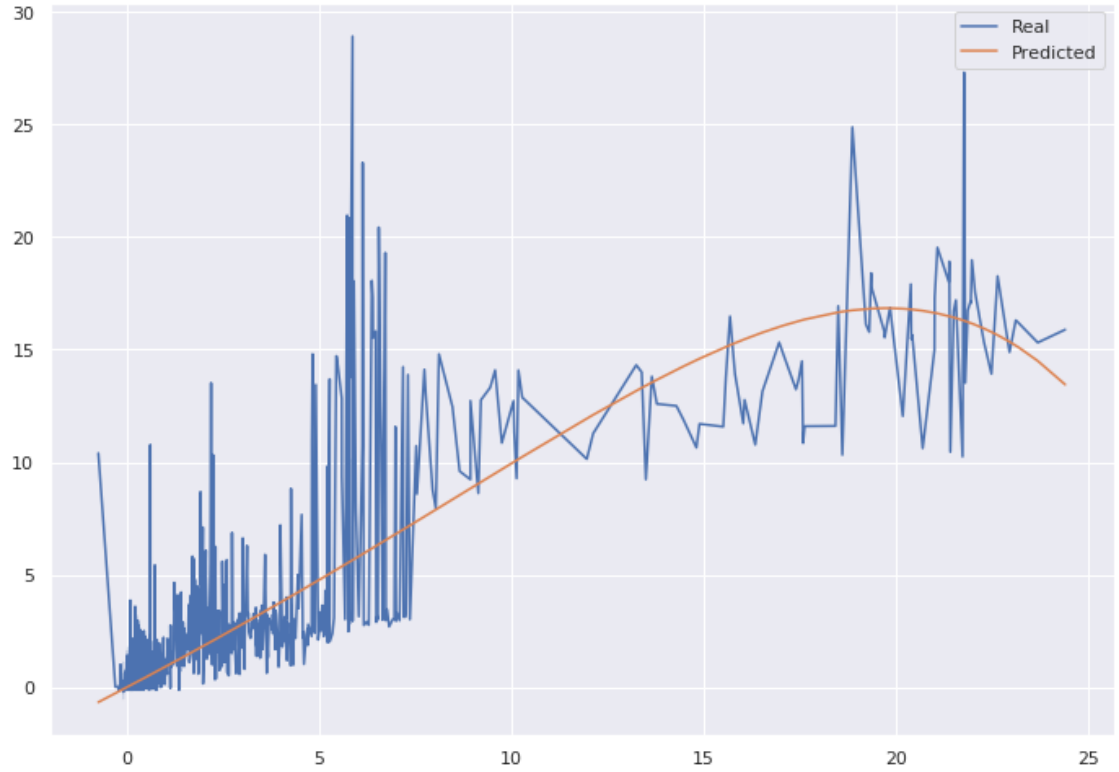
En taronja es mostra com donantli una mica d'importancia al regularitzador hem aconseguit reduïr el overfitting que feia tant rapid allargant-lo a varies iteracions i donant-li un aprenentatge mes consistent i amb un learning rate mes alt; per tant fent-ho mes rapid.

1.24.3 4.- Té sentit el model (polinomial) trobat quan es visualitza sobre les dades?

[63]:
```
info3 = {'x_axe': [] , 'y_axe': [] , 'TYPE':[]}
poly = lambda x,model: model.weights @ x
for x_axe,y_axe in␣
 ↪zip(normalitzador_de_dades(dataT),normalitzador_de_dades(dataV.
 ↪reshape(-1,1)).reshape(1,-1).flatten()):
  info3['x_axe'].append(x_axe[1])
  info3['TYPE'].append('Real')
  info3['y_axe'].append(y_axe)

  info3['x_axe'].append(x_axe[1])
  info3['TYPE'].append('Predicted')
  info3['y_axe'].append(poly(x_axe,reg))
```

[64]:
```
sns.lineplot(data=info3,x='x_axe',y='y_axe',hue='TYPE')
```

[64]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdf8bb325d0>
```



Podem veure com la corva de grau 5 s'adapta prou a les dades donades

[ ]: